An Argus Specialist Publication

Computing today

FEBRUARY 1984
85p

MACHINE CODE MADE EASIER
Monitor review for the Spectrum

MULTI-TASKING ZX81-FORTH – Can it do the job?

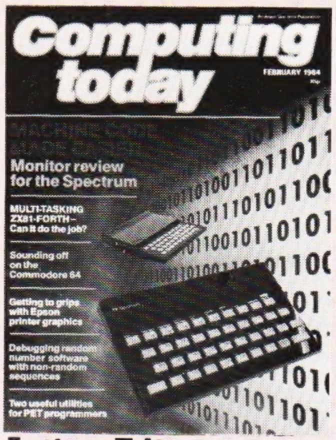
Sounding off on the Commodore 64

Getting to grips with Epson printer graphics

Debugging random number software with non-random sequences

Two useful utilities for PET programmers

ANIROG SOFTWARE
COMPUTER GAMES OF TOMORROW AVAILABLE
NOW!
AS SEEN ON T.V.!
HEXPERT
MOON BUGGY
SKRAMBLE
3D TIME TREK
COMMODORE 64
NEW
GALAXY
AVOID CAPTURE BY THE ALIEN MOTHER SHIPS TRACTOR BEAM AS THE FIGHTERS DIVE BOMB YOU. 100 SCREENS WITH A TWO PLAYER OPTION K.B./J.S. £7.95
KONG K.B./J.S. £7.95
4 SCREENS WITH BRILLIANT ANIMATED GRAPHICS
SKRAMBLE J.S. £7.95
6 SECTORS TO TEST YOUR SKILL
HEXPERT J.S. £7.95
TRAIN BERT TO BE HEXPERT ON THIS 3D HEXAGONAL PYRAMID
MOON BUGGY J.S. £7.95
MANOEUVRE YOUR PATROL CRAFT OVER GIANT POT HOLES AS YOU DEFEND THE MOON BUGGY FROM ALIEN ATTACK
FROG RUN K.B./J.S. £5.95
A POPULAR ARCADE GAME. FUN FOR THE WHOLE FAMILY
3D TIME TREK K.B./J.S. £5.95
SPECTACULAR 3D GRAPHICS STAR TREK GAME
DUNGEONS K.B. £6.95
ENTER THE REALMS OF FANTASY IN THIS ROLE PLAYING GAME
DARK DUNGEONS K.B. £6.95
2ND IN THE SERIES OF FOUR DEFINITELY NOT FOR THE FAINT HEARTED
NEW APPROVED GAMES FOR
THE STACK LIGHT RIFLE
INDIAN ATTACK £5.95
COSMIC COMMANDO £5.95
VENGEANCE OF ZENO £5.95
24 HR. CREDIT CARD SALES HORLEY (02934) 6083
PAYMENT BY CHEQUE, P.O., ACCESS/VISA
8 HIGH STREET HORLEY, SURREY.
Overseas 50p post & packaging
TRADE ENQUIRIES WELCOME
29, West Hill, Dartford, Kent. (0322) 92513/8



Acting Editor:
Peter Green
Editorial Assistant:
Jamie Clary
Advertisement Manager
Malcolm Wynd
Sales Executive
David Poulter
Advertisement Copy Control:
Sue Couchman,
Ann McDermott
Managing Editor:
Ron Harris BSc
Chief Executive:
T.J. Connell
Origination by Design International.

ABC Member of the Audit Bureau of Circulation

ISSN 0142-7210

Computing Today is normally published on the second Friday in the month preceding cover date. Distributed by: Argus Press Sales & Distribution Ltd, 12-18 Paul Street, London EC2A 4JS. 01-247 8233. Printed by: Alabaster Passmore & Sons Ltd, Maidstone, Kent.

Subscription Rates : UK £13.30 including postage. Airmail and other rates upon application to Computing Today Subscriptions Department, 513 London Rd, Thornton Heath, Surrey CR4 6AR.

Computing Today is constantly on the look-out for well written articles and programs. If you think that your efforts meet our standards, please feel free to submit your work to us for consideration.

All material should be typed. Any programs submitted must be listed (cassette tapes and discs will not be accepted) and should be accompanied by sufficient documentation to enable their implementation. Please enclose an SAE if you want your manuscript returned, all submissions will be acknowledged. Any published work will be paid for.

All work for consideration should be sent to the Editor at our Charing Cross Road address.

CONTENTS

VOL 5 NO 12 FEBRUARY 1984

EDITORIAL & ADVERTISEMENT OFFICE
No. 1, Golden Square, London WC1 3AB.
Telephone 01-437 0626. Telex 8811896.

GET YOUR HANDS ON ONE...

The

Spectravision Quickshot deluxe joystick (BBC compatible)

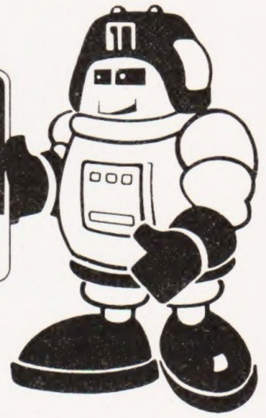
Now, Microstyle offer you the chance to use a joystick where joysticks have never been used before! Operating directly through the existing user-ports on your BBC model B microcomputer and available for use with all Acornsoft and most other keyboard controlled games even Planetoids — these joysticks include a machine-code 'driver program' on tape, which converts keyboard commands to remote control operation for maximum enjoyment.

only £19.95 inc VAT

for mail order please add 75p P&P

MicroStyle

THE HOME COMPUTER PEOPLE



The Aylesbury Computer Centre
52 Friar's Sq., Aylesbury.
Telephone: Aylesbury (0296) 5124

The Bath Computer Centre
29 Belvedere, Lansdown Road, Bath.
Telephone: Bath (0225) 334659

The Daventry Computer Centre
67 High St., Daventry.
Telephone: Daventry (03272) 78058

The Newbury Computer Centre
47 Cheap Street, Newbury.
Telephone: Newbury (0635) 41929

Please Note: While Microstyle will endeavour to maintain sufficient stocks of items currently or previously advertised we regret that all items are offered for sale subject to availability from manufacturers.

...BEFORE THEY DO!

CONSUMER NEWS

SV ARRIVAL

The powerful Spectravideo range of personal computers, currently enjoying a huge sales success in the US, is now available for the first time in the UK. Distributed exclusively by CK Computers of Weston-super-Mare, Avon, the Spectravideo range comprises two models, the SV318 and SV-328.

A major selling feature of both Spectravideo machines is their adoption of the MSX specification, which is expected to become the industry standard for home computers. MSX uses Microsoft BASIC as its resident interpreter and gives access to a wealth of computer games and other personal computer software. In addition, the SV-318 and the SV-328 are also CP/M compatible, which means that as users' demands become more sophisticated they can take advantage of the comprehensive range of software programs written for CP/M such as Wordstar and Visicalc.

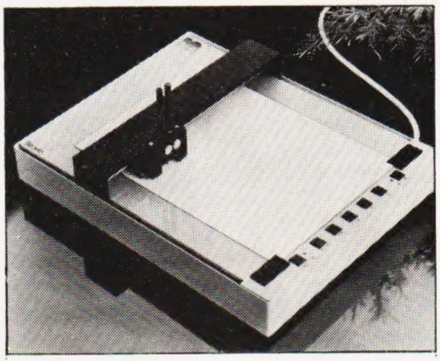
A further very important factor is that Spectravideo have a range of 15 add-on peripherals including RAM expansion cards, floppy disc drive, graphics tablet and printer. All of these are available now from CK Computers, which makes a nice change from other makes.

The SV-318 at £173 complete with data cassette is extremely competitively priced for a personal computer with these features and a memory capability of 32K RAM expandable to 144K RAM and 32K ROM expandable to 96K ROM. The SV-318 also has a joystick built into the keyboard, another unique feature in this price range.

For the professional businessman the model SV-328 (priced at £239 complete with data cassette, 80K of RAM expandable to 256K and 48K ROM expandable to 96K) represents a long term investment which allows memory to grow as the users' requirements increase. Main features include an 87-key keyboard, 80-column display and numeric keypad, and 256 by 192 pixel screen resolution. CK Computers Ltd are at 6 Devonia House, High Street, Worle, Weston-super-Mare, Avon BS22 0JR (telephone 0934 516246). **Computing Today** will shortly be receiving a Spectravideo computer for review. Watch these pages!

DAM CLEVER

Linear Graphics Ltd have announced a £450 plotter for personal computers that employs new linear motor and optical feedback technology to achieve repeatable 0.2mm accuracy over the whole of the plotting area. Known as the Beaver, the plotter is the result of an intensive development and production engineering program at Linear Graphics that has spanned the past year or so.



The Beaver has a Centronics interface as standard (RS232 optional), and can therefore be used with practically any computer. Special software is available, at additional cost, for the BBC models A and B, and Apple II and IIe machines. This software is called 'Interceptor' and has been developed concurrently with the plotter by Linear Graphics.

Interceptor is a powerful routine that intercepts graphic commands for plotting and drawing from BASIC and routes them either to the screen or the plotter as required by the user. As a result, graphics programs already written for the BBC or Apple PCs can run with the Beaver with little or no modification.

The combination of a universal pen holder and a PEN CHANGE command allows most popular 'Roller Ball' or fibre-tipped pens to be used. The PEN CHANGE command causes the pen holder to move to a pen change position on the bed, making it very easy for the user to set the pen at the correct height in the holder.

The Beaver is a flat bed machine with a plotting range of 190 x 272 mm (A4) and will draw on paper, transparencies for overhead projection or even on the backs of envelopes! The paper, or transparency, is held in position by magnetic rubber strips. Accuracy is better than 0.2 mm regardless of the distance moved.

At the right-hand edge of the plotting bed there are a number of switches for manual control of the plotter. These include North, South, East, West, Pen Down, Pen Up, and Line/Local.

The Beaver measures 302 x 381 x 97 mm and weighs in at only 8 kg. Further information can be obtained from Linear Graphics ltd, 34a Brook Road, Rayleigh Weir Industrial Estate, Rayleigh, Essex (telephone 0268 741322).

THE SEEING EYE

A low cost video-camera-to-computer interface aimed at the educational and semi-professional user has recently been launched by Educational Electronics. The interface accepts signals from a variety of sources such as a video camera, VHS player and video disc. It can digitise an image with a resolution of 220 (horizontal) by 312 (vertical) pixels with 64 levels of grey. The unit has wide applications in the fields of art, design, science, robotics and technology.

The low cost of the Video Interface has been made possible by replacing many functions usually performed by hardware with appropriate software. This also greatly

enhances flexibility as the software can enable various parts of the image to be selectively scanned, giving more detail or detecting rapid movement in certain defined areas. Trade-offs can also be made between computer memory size, number of pixels scanned and the number of bits per pixel (representing the intensity) stored in memory. The information can then be displayed on a monitor, saved to disc or processed to extract specific information (such as area and perimeter analysis, shape recognition etc.) In addition the ability to attribute a specific colour to a particular intensity (ie the use of 'false colour') can be used to highlight certain features of the image.

The Interface can be used on virtually any micro with a user port. The unit comes complete with mains power supply, extensive documentation, software support and a connecting lead for the BBC Model B, RML 380/480Z or Apple user port. The cost is £174 (excluding VAT) and further details can be obtained from Educational Electronics, 30 Lake Street, Leighton Buzzard, Beds LU7 8RX (telephone 0525 373666).

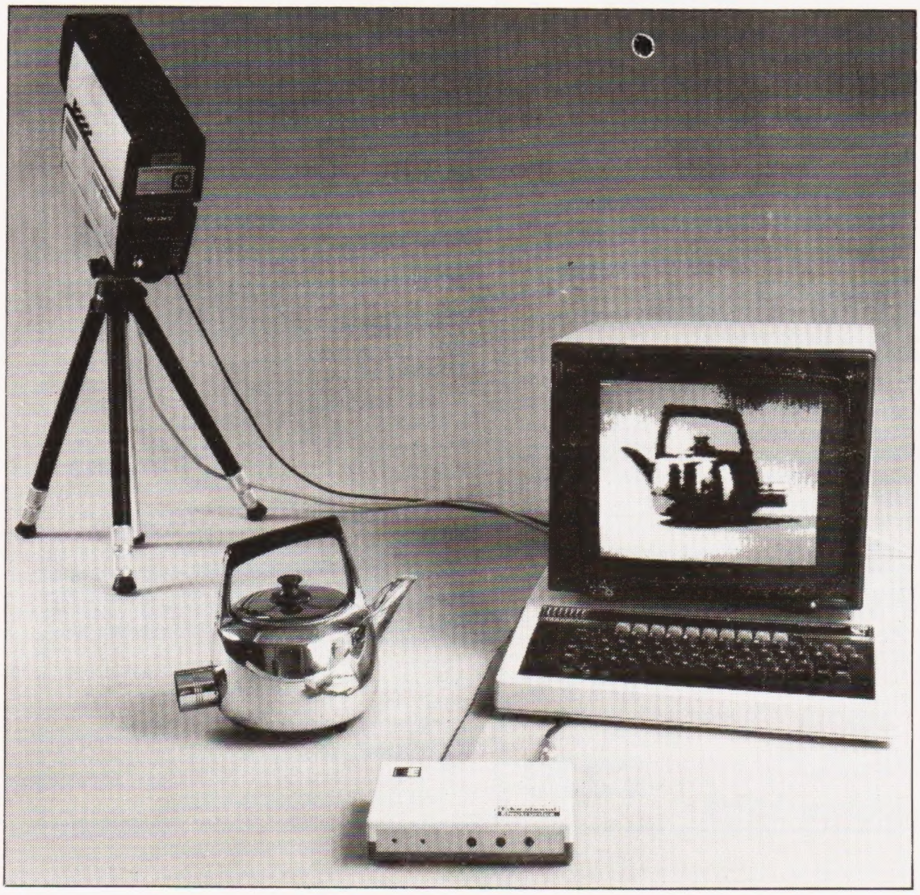
MINI-MODEM

Tech-Nel Data Products Limited has launched a low cost ultra-miniature short-range modem, the SRM-6, which costs about one quarter of the price of an equivalent conventional modem and measures just 4.5 x 2.2

x 10.6 centimetres. The SRM-6 is ideal for short haul data transmission, up to 25 kilometres, and needs no AC power supply or batteries. It is therefore extremely suitable for use in large office and factory complexes.

The SRM-6 plugs directly into standard CCITT V-24/RS-232C terminals or computer digital interface connectors. It takes its power from signals emitted by the terminals and from transmit and receive signals, so that no external power source is required. Data is transmitted in full duplex and for wire asynchronous modes over unconditioned telephone lines at any rate up to 19,200 bps.

SRM-6 modems are available by mail order from Tech-Nel in bubble-packed sets of two at prices from £140 per pair, or even less from quantity purchases. Tech-Nel Data Products Ltd are at 8 Haslemere Way, Banbury, Oxon OX16 8TY (telephone 0295 65781).



DON'T WASTE YOUR TIME

Printing out directly from a computer to a printer ties up the computer, often when you most need it. The new compact Microbuffer from Inmac can store up to 64K bytes (or approximately 30-40 pages of A4 text) in its memory as fast as the computer can dump it. This is then fed into the printer at its slower rate, completely freeing the computer to do other operations.

The Microbuffer is compatible with most microcomputers including IBM PC, Apple, TRS-80 and with leading manufacturer's printers such as Epson, NEC, Diablo, C. Itoh and Centronics. It can also be used with most makes of plotters and modems. No modifications are required to the existing software and connection is by standard plugs and cable.

There are two versions of the Microbuffer available which both have a data transfer rate of 4000 cps and cost £225 each (including an AC adaptor). The parallel version comes with a 2 m buffer-to-printer cable. The serial version uses standard RS232 (V24) cable interfaces, has two handshake modes, nine baud rates and a bypass feature for instant printer access.

As with all Inmac products, the Microbuffer comes with a full one year guarantee and is available on a 30 day risk-free trial period and with next day delivery. Full details can be found in Inmac's full colour catalogue which is available free from Inmac (UK) Limited, Davy Road, Astmoor, Runcorn, Cheshire WA7 1QF (telephone: 09285 67551).

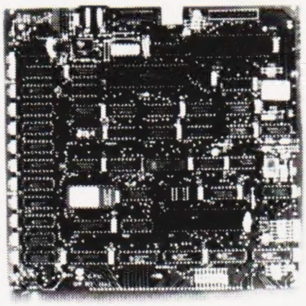
PODS DOWN IN PRICE

Oxford Computer Systems have reduced the price of Interpod — their interface system for the Commodore 64 — to £99.95.

Interpod is an intelligent interface that provides the Commodore 64 with both RS232 and IEEE interface capabilities. Thus users of Commodore's latest home computers are able to take advantage of the wide range of peripherals such as dual disc drives and daisy-wheel printers, and hence extend the capabilities of their system in a low-cost and powerful manner.

Interpod is available from Oxford Computer Systems, from the UK network of Commodore dealers or from the world-wide network of dealers and distributors for £99.95. For further information please contact Oxford Computer Systems Limited, Hensington Road, Woodstock, Oxford (telephone 0993 812700).

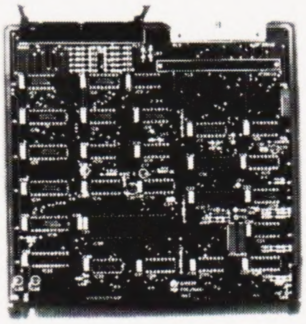
MicroValue 80-BUS MULTIBOARDS



GM813—CPU/64K RAM Board

* 4MHz Z80A C.P.U.
* 64K Dynamic RAM
* RS232 Interface
* Two 8-Bit I/O Ports
* Cassette Interface
* Extended & Page Addressing Modes
* CP/M Compatible Monitor

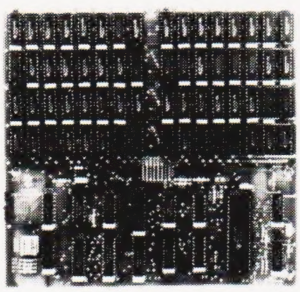
£225



GM829—Disk Controller Board

* Up To 4 Mixed 5.25 & 8" Drives
* SASI Hard Disk Interface
* Single & Double Density Operation
* Single & Double Sided Drive Support
* Supports 48 and 96 TPI Drives

£145

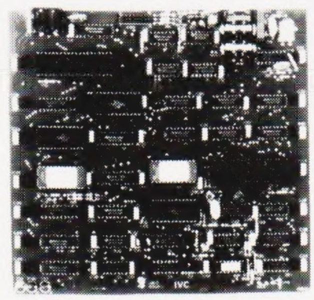


GM833—RAM-DISK Board

* Virtual Disk Operation
* 512K Dynamic RAM
* Port Mapped For Easy Interface Software
* Over 10 Times Faster Than a Floppy Disk

£450

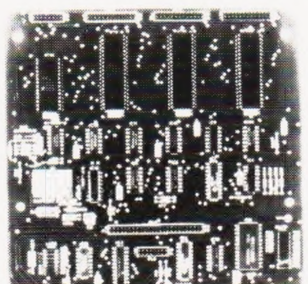
Please note: This board cannot be used as a conventional RAM board



GM812—Video Controller Board

* 80 Characters×25 Line Display Format
* On-board Z80A Microprocessor
* Buffered Keyboard Input
* Programmable Character Generator
* 160×75 Pixel Graphics
* Light Pen Input

£125



GM816—Multi I/O Board

* 6 I/O Ports
* 4 Counter/Timer Channels
* On-Board Real Time Clock
* Battery Backup
* Further Expansion Capability

£125

PLUTO—Colour Graphics Board

* On-Board 16 Bit Microprocessor
* 640×576 Bit Mapped Display
* 192K Of Dual Ported RAM
* Comprehensive On-Board Software

£499

CLIMAX—Colour Graphics Board

* 256×256 Pixel Display
* 16 Colours
* Ultra-fast Vector & Character Generation
* Light Pen Input
* UHF or RGB Outputs

UHF Version **£199**

UHF & RGB Version **£220**

GM811—CPU Board

* 4MHz Z80A CPU
* 4 'Bytewide' Memory Sockets
* 2×8-Bit Input/Output Ports
* 8 Bit Input Port
* RS232 Serial Interface
* Cassette Recorder Interface

£125

GM803—EPROM/ROM Board

* Up to 40K of Firmware
* 2708 or 2716 EPROMS
* Page Mode Operations

£65

GM802—64K RAM Board

* 64K Dynamic RAM
* 4MHz Operation
* RAM Disable Function
* Page Mode Operation

£125

MP826—Static RAM Board

* 32K Static RAM
* Battery Backup
* Page Mode Operation

£225

EV814—IEEE488 (GPIB) Controller

* Cost Effective Controller
* Comprehensive Software Supplied
* Full Implementation
* Easy To Use

£140

GM827—87 Key Keyboard

* User Definable Function Keys
* Numeric Keypad
* Cursor Control Keys

£85

GM839—Prototyping Board

* Fibreglass P.C.B.
* 80-BUS Signal Identification
* High Density IC Capability

£12.50

MULTIBOARD COMPUTERS

Gemini Microcomputers

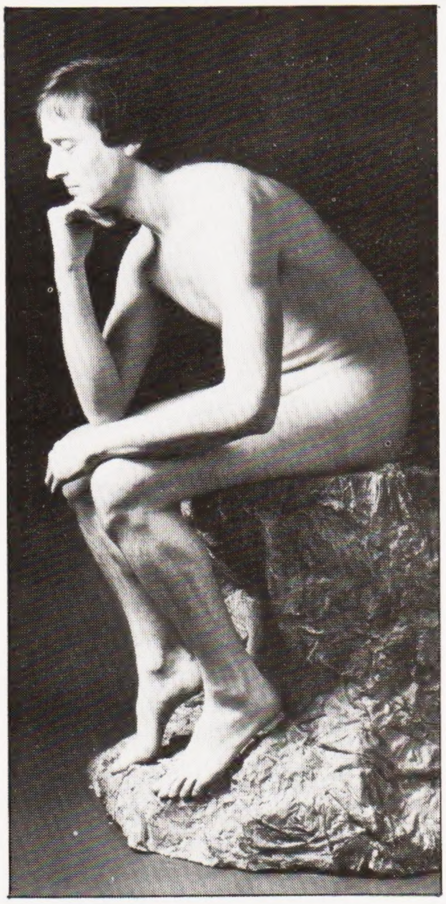
All the boards and components in the 80-BUS range are fully compatible and offer a very flexible and cost effective solution to your computer needs. For further information about the 80-BUS range contact your nearest MICROVALUE dealer.

SOFT WARES

BRAIN(STORM), NOT BRAWN

This magnificent specimen of humanity is Mike Liardet, and he is the co-author of BrainStorm, an 'ideas processor' in the same way that a word processor processes words and a spreadsheet processes numbers. I suppose that makes him a bit cleverer than the rest of us mortals, but frankly I feel that anyone who volunteers for a photograph like this isn't dealing with a full deck, if you know what I mean. **Computing Today** is, at this very moment, preparing a review of this software package, and I suspect other people may be working a report of Mr. Liardet's mental state.

That whirring noise is Rodin spinning in his grave.



MORE DISCS FROM MOLIMERX

Molimerx are pleased to announce that they have been appointed sole distributors for CP/M 2.2 for the Tandy Model 4 outside the United States. This version from Montezuma Micro is a full CP/M with the original Digital Research utilities, plus a number of others.

Of particular importance is a file which enables this CP/M to interface with 22 other formats of CP/M. Additionally, a utility is included which converts an area of RAM into a pseudo disc drive. Communications and terminal software are also included. The present Tandy owner will be particularly pleased that an extremely easy Format and Backup utility is also included.

This version of CP/M is intended to fill the gap in time pending the issue of the CP/M being written for Tandy by Digital Research. If, when that is released, it is found to be superior to the Montezuma version, then the latter will be discontinued. On the other hand if, as it rather appears at the moment, the Montezuma is superior to the Tandy, then Molimerx will stock both.

Molimerx have also concluded a licensing agreement with Logical Systems Incorporated of Milwaukee, United States, the publishers of LDOS, the disc operating system for the Tandy Model I, Model III and Model 4 machines. This new agreement will enable Molimerx to manufacture in the United Kingdom products to be sold worldwide outside of the continental USA.

The agreement will also enable Molimerx to make a major breakthrough in pricing. Henceforth all LSI products, including LDOS, smalLDOS, The Basic Answer and other well-known packages will be sold in the United Kingdom to the end user at the pound equivalent of the US retail price. In other words, there will be no reflection in the new pricing schedules of Customs duty or shipping.

It is believed that this is the first time in the software industry that the price in the United Kingdom will be the same as that in the USA.

Molimerx Ltd is at 1 Buckhurst Road, Town Hall Square, Bexhill-on-Sea, East Sussex (telephone 0424 220391/223636).

GEMINI GRAPHICS

Henry's of Edgware Road has extended its range of available software packages with the introduction of IVC HI-RES, a program which provides pseudo high resolution graphics on 640 by 250 matrix.

IVC HE-RES has been specially written for the Gemini Multiboard computers, the Gemini Galaxy range, the Quantum 2000, the Kenilworth Personal Computer and CP/M based Nascoms fitted with the Gemini GM 812 IVC. It achieves pseudo high resolution graphics by reprogramming the video control processor and mapping the programmable set generator to the screen.

Available from Henry's at £15 plus VAT, IVC HI-RES provides the following commands: Select mode (48 or 80 column); clear graphics screen; select decimal or binary coordinates; set, reset, invert and test point X, Y; line set, line reset and line invert line X, Y to X1, Y1.

Henry's can be found at 404-406 Edgware Road, London W2 1ED (telephone (01-402 6822).

MICRODRIVE BUDGETTING

With the recent introduction of the ZX Microdrive you now have the ability to load the Cash Controller program and make an entry in around 90 seconds. This upgrades the performance of the 48K Spectrum for more serious roles.

Richard Shepherd Software have just released a professional-style Cash Controller program (believed to be the first) that has a "SAVE-to-ZX Microdrive" option in the main menu. The obvious advantage of this Microdrive capability is that it allows the user to SAVE the program onto a blank Microdrive cartridge when supplies are more readily available.

This home budgeting and banking system handles up to 400 transactions which can be automatically allocated against 16 selected budget headings such as Rates, Gas, Tax and so on. The program also gives statements on demand.

Cash Controller for the 48K Spectrum with ZX Microdrive compatibility costs £9.95 and is now available by mail order, telephone credit card order or from most leading computer stores. Contact Richard Shepherd Software, Elm House, 23-25 Elmshott Lane, Cippenham, Slough, Berks (telephone 06286 63531).

LEARN WITH GRIFFIN

Griffin Software, part of Griffin & George, has launched a new range of educational programs for use on home computers, for young children in the 4-9 age bracket. There are, initially, six programs taking the form of instructional booklets for parents plus computer software tapes attractively packaged and colour-coded.

The new range of Griffin Software children's educational programs is for use initially on two types of home computer — the Sinclair ZX Spectrum 48K and BBC Model B 32K microcomputers — which together constitute some 60% of the total UK home computer market. The educational software will, however, be progressively extended to other home computers as appropriate.

Four of the new home computer programs for 4-9 year olds are available from Smiths, Boots and other leading retail outlets now — 'Wordspell' (spelling); 'Getset' (numbers); 'Numberfun' (addition and subtration); 'Tablesums' (multiplication), while the other two programs — 'Fairshare' (division) and 'Wordgames' (more advanced spelling) — will both be in the shops by the end of November.

The six programs, with colour-coded packaging — blue for software for use on the Sinclair ZX Spectrum and green for the BBC Model B microcomputers — are priced at £7.99 for Spectrum and £9.95 for BBC Model B.

ALLIGATA DATA

Flexibase, Alligatacalc and Scribe II are three home/business utilities for the BBC Model B now available from Alligata Software, of Sheffield.

Flexibase is, as the name suggests, a flexible master database. Available on tape and disc, Flexibase enables users to extend the number of records they can hold by selecting the number of fields in each record and then the number of characters in each field. Output is to either screen or printer. All records can be sorted alphabetically on the first field, with a secondary sort on any of the first 10 fields in preferred order. (RRP is £9.95 on tape; £13.95 on disc).

Alligatacalc is a simplified financial/accounting package designed specifically for the BBC to handle the following tasks: cash flow forecasting: budgetary control; estimating; price lists; discount structures; profit and loss accounts; profitability charts; home finance control; shopping lists. The program will automatically calculate any changes in detail input and instantly correct affected totals, which means that constant updating is simple and fast. (RRP is £9.95).

Scribe II is a professional word processor for the BBC Model B, fully compatible with all versions of the operating system and able to be used parallel or serial printer. The program is simple to use but very powerful, and handles up to 600 lines of text (about two A4 sheets). The main features include menu drive; block insert/replace/delete; 80 characters per line display on screen; adjustable column width; save/load files to tape/disc; print as formatted or unformatted text; user-defined key operation for easy use. (RRP is £9.95 on tape; £14.95 on disc).

For more information contact Alligata Software, 178 West Street, Sheffield S1 4ET.

MICROWRITING FOR PETS

Commodore PET users can now communicate with the Microwriter — the portable hand-held wordprocessor with a unique and extremely simple to use keyboard of just six keys. Microcomputer Services, an appointed Microwriting Centre, has developed the software program 'Speakeasy', which allows two-way transfer of text between PETs and Microwriters.

Now PET microcomputer users can transfer text to their data discs for storage, merging of files or for printing out at a convenient moment. Documents can also be retrieved from the PET and entered into the Microwriter's memory for reference, updating or amendment. The Microwriter can also be used in a networked environment. An interface lead, enabling communciation between the PET, which has IEEE connectors, and the Microwriter's in-built RS232 interface is available from Microcomputer Services.

The 'Speakeasy' program is available from Microcomputer Services, priced at £140. It will also be available soon from other Microwriting centres around the UK. Details are available from Osman Ismail or Leslie Bird at MCS, telephone number 01-802 0019, or 01-809 3896.

DATA GENIE FOR SPECTRUM

Following on the launch of Magpie for the Commodore 64, Audiogenic have now announced Data Genie Data Genie is an easy-to-use database and record retrieval package that allows users of the Sinclair Spectrum to organise their own records in their own unique manner and to recall them under a wide range of parameters.

The package is controlled through the novel method of 'pop-up' menus which are managed by just three keys. The user selects the required option from each menu by raising or lowering a cursor line. Once the cursor is over the required option, a third key automatically pulls in the menu relating to the option, overlaying the new menu on the previous menu. The user is thus able to follow clearly the steps taken in building up the database. As a menu element is selected, Data Genie automatically writes the relevent part of the program.

Data Genie is supplied on cassette for the Sinclair 48K Spectrum at a cost of £9.95 and is available direct from Audiogenic or the nationwide dealer network. Audiogenic are at PO Box 88, Reading, Berks RG1 2SN.

NEXT MONTH

computing today

MARCH ISSUE ON SALE FEBRUARY 10th

POKER

Following the ZX81 version of Backgammon in the September issue, we now offer you BBC Poker. Written for the 32K BBC Micro with any operating system, this game will shuffle the deck, deal and bet with you, and all without the danger of losing your shirt. The only disadvantage is that you cannot read the expression on the computer's face to find out if it's bluffing! User-defined graphics allow your hand to be displayed on-screen as a set of cards, and if you do run into trouble the computer will even advance a loan of £1000. Bring the casino into your own home with the March **Computing Today.**

EASYCODE

There are some difficulties attached to any series of articles that attempt to teach machine code. For a start, there's a different instruction set for each type of microprocessor. Then the type of microprocessor available depends on which computer you have: not all readers will have the same processor to hand.

We've overcome the problems by inventing our own microprocessor! Easycode uses a simulated microprocessor with 100 'memory locations' available, and using this model we can teach the general principles of machine code programming. The simulation will run on any home computer which supports BASIC and a TV display.

ZX SPECTRUM PRINTER INTERFACE

The Sinclair printer is nice and cheap, but the results are nasty and cheap-looking. It would be better to use a good-quality printer but there are a number of problems: the Spectrum has no Centronics interface and it puts out the wrong codes anyway. Next month we'll be publishing a combined hardware and software project to allow printing on a Centronics device: you can use either a simple DIY interface or the ready-built product from Kempston Electronics, and suitable software will be given for both types.

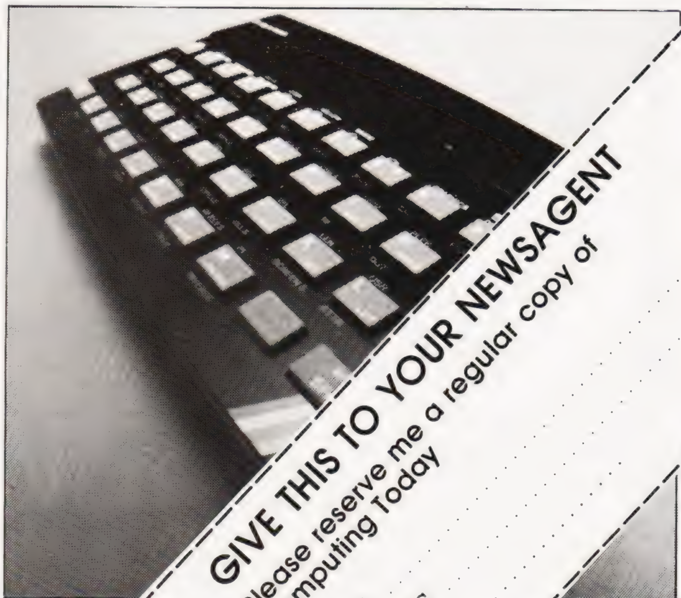
Articles described here are in an advanced state of preparation but circumstances may dictate changes to the final contents.

GENIE UTILITIES

This short piece of software provides a wealth of extra facilities for the Genie owner. USR calls with multiple parameters, a VDU-type command for easy output of ASCII characters to the display, and conversions between various bases — it's all packed into only 334 bytes of machine code. Don't miss the March issue of **Computing Today.**

SCOPE FOR IMPROVEMENT

The Spectrum is capable of some very advanced graphics effects, as evidenced by the latest in commercial software for the machine. Unfortunately it's been necessary to get your hands dirty with machine code if reasonably-paced arcade games are to be attempted. Until now, that is. SCOPE (Simple Compilation of Plain English) is a collection of just 31 commands that allow anyone to design a game and make it run at machine code speeds. We'll be reviewing this piece of software in the March issue.



GIVE THIS TO YOUR NEWSAGENT

Please reserve me a regular copy of Computing Today

NAME ....

ADDRESS ....

POSTCODE ....

Software News

INNOVATIVE
TRS 80-GENIE SOFTWARE

from the professionals

NEWS FLASH!

Molimerx and Logical Systems of Milwaukee, U.S.A., have joined forces to bring their customers a lower costing product and faster and more efficient service.

From January 1984 all LDOS 5.1.× Logical Systems products, together with some of the LDOS (TRSDOS) 6.× products, will be available from Molimerx at the pound equivalent of the U.S. Dollar retail price. In other words, for the first time the considerable range of products of Logical Systems Incorporated will be available to the end user in the United Kingdom at the price at which the American customer can buy it in the U.S.A. All support for these products is being shifted to England, so that as from 1st January, U.K. customers can have the benefit of this important line, exactly as if it had been written and produced over here.

Adjustments for the exchange rate will be made every six months or so. We are starting with the present exchange rate of 1.48. After VAT is added this scheme results in the price schedule (plus P&P) that follows:

Name	Brief Description	Previous Selling Price	Present Selling Price
DISK/DISK	Convert a disk file to another "disk drive"!	£n/a	£ 76.94
FED II 5.1	All purpose disk file editor	£ 27.60	£ 27.60
FILE MANAGER 5.1	Utility for mass manipulation of files	£ 33.40	£ 30.30
FILE MANAGER 6.0	Utility for mass manipulation of files	£ 33.40	£ 30.30
FILTER PACKAGE 1	Filters to enhance LDOS	£ 22.71	£ 22.54
FILTER PACKAGE 2	Filters to enhance LDOS	£ 22.71	£ 22.54
FIX DISK	A collection of patches for LDOS	£ 13.80	£ 7.76
HELP 5.1	LDOS and LBASIC help	£ 17.25	£ 14.78
HELP GENERATOR 5.1	Create your own HELP files	£ 33.35	£ 33.35
HELP TEXT SOURCE	Source files for creating main HELP files	£ 17.25	£ 14.78
INVENTORY	An aid to inventory tracking	£ 82.50	£ 76.94
I/O MONITOR	Disk I/O error intercept utility for LDOS	£ 22.43	£ 14.78
LDOS 5.1.×	New generation disk operating system	£105.80	£100.28
LDOS TECH. HELP	Technical help for LDOS	£ 20.70	£ 20.70
LED	Screen orientated text editor	£ 21.85	£ 21.85
MAIL/FILE II	A mailing list database manager	£ 82.50	£ 76.94
MEMDISK	Additional disk type storage	£ 28.69	£ 22.54
QUIZ MASTER	Questions and Answers — Master includes general	£ 33.40	£ 30.30
QM GEOGRAPHY	Questions and Answers — Geography Requires Quiz Master	£ 17.25	£ 14.78
QM MATH	Questions and Answers — Maths Requires Quiz Master	£ 17.25	£ 14.78
Smal-LDOS	Miniature of the original LDOS	£ 43.70	£ 43.70
T.B.A. 5.1	Basic text processing utility	£ 51.75	£ 51.75
T.B.A. 6.0	Basic text processing utility	£ 57.50	£ 57.50
ULTRA TREK	Space Wargame	£n/a	£ 11.87
UTILITY DISK I	LDOS enhancement package	£ 33.46	£ 30.30

TEL: [0424] 220391/223636

TELEX 86736 SOTEX G

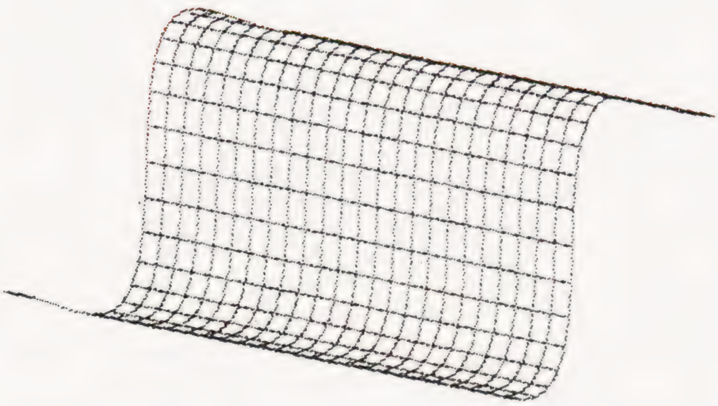
MOLIMERX LTD

A J HARDING (MOLIMERX)

1 BUCKHURST ROAD, TOWN HALL SQUARE, BEXHILL-ON-SEA, EAST SUSSEX.

James I. Bartholomew

USING EPSON MX-80 GRAPHICS



The Epson MX-printer is a popular and versatile beast which is capable of a great deal more than just listing your programs. In this article we show you how to make the most of graphics.

When I originally decided to buy a printer it was to help me in the development of programs, to print out results and to use in writing reports and letters. I chose the Epson MX-80 because it was a good, clear typeface and also because it had a dot graphics capability which I thought would compensate for the lack of high resolution graphics on my TRS-80.

When you look at advertising brochures and reviews of printers they usually illustrate the graphics capability by contour maps of some mathematical function and pictures of pretty ladies made up from individual dots. There is very little information, however, either in books or magazines on how to produce these pictures for yourself at home. A digital converter would be required to change a photograph into data for a computer, but mathematical functions can be graphed quite easily.

A favorite function of mine is Z=10★SIN(X)/X, which crops up frequently in science and engineering and has a pleasant appearance. If you wanted to plot this on the VDU you would use a program such as Listing 1 which takes each value of X, calculates the corresponding value of Z, scales the coordinates and displays the point on the screen.

This direct approach cannot be used with the MX-80 as it makes use of the random access characteristic of the VDU; each point can be SET or RESET in any order. With a printer you must start at the top and work down. Once it has advanced to print a lower point, it cannot go back up again.

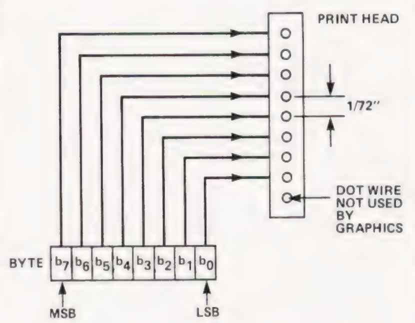


Fig. 1 The association between the data bits and dot wires.

It is necessary, therefore, to calculate each point first, store the results and finally print them. Before we do this, let's remind ourselves of how the MX-80 prints dot graphics.

EPSON GRAPHICS MODE

To enter the normal density bit image mode, the computer must send an ESC K instruction to the printer. ESC is CHR$(27) and K is CHR$(75), sent to the printer as two consecutive bytes, one of 27 and the next of 75.

The next two bytes (n1 and n2) sent to the MX-80 tell it how many bytes (n) of bit image data follow; up to a maximum of 480, the total number of dot positions in a line. These are in order (least significant byte), (most significant byte) and the total number of graphics bytes is given by

```
n = n1 + 256*n2
```

For example, if you want to print 110 bytes of bit image data then n1=110 and n2=0, or to send 310 bytes then n1=54 and n2=1.

Once ESC K n1 n2 has been sent to the printer, the next n bytes will be interpreted as image data and not as characters or printer controls.

Double density graphics are controlled in the same manner by sending ESC L (27 then 76) to the printer, and n1+256★n2 can total up to 960 because dot spacing is halved and there are twice as many in a line.

The print head of the MX-80 contains nine wires one above the other, each capable of printing a single dot. All characters are made up of combinations of these dots. When you enter bit image mode, only the upper eight wires are active, and each bit of a graphics byte controls one wire as shown in Fig. 1.

The most significant bit controls the top wire and the least significant bit the bottom. So, a byte of 255 will fire all eight wires to print eight dots, 15 will print the lower four dots and 0 will print none.

PLOTTING THE FUNCTIONS

Let us now look at two methods of plotting the function Z=10★SIN(X)/X. First of all, we can store all the values of Z in an array and then calculate which should be printed on each row. Or, secondly, we can plot each point into a buffer and then dump the buffer out to the printer.

In the first method, for the range of values of X that we want to use we calculate each value of Z and store it in an array Z(X). Then we scan the array for each value of Z, starting at the highest, and print a dot at the appropriate X position. This is done by Listing 2 and the result is shown in Fig. 2.

If we decide to calculate for values of X between −15 and 15 in

```
99  REM ** Clear the screen **
100 CLS
109 REM ** Calculate function **
110 FOR X=-15 TO 15 STEP .25
120    IF X=0 THEN Z=10: GOTO 140
130    Z=10*SIN(X)/X
139    REM ** Scale coordinates for display **
140    XD=4*X+64
150    ZD=36-3*Z
159    REM ** Display point **
160    SET(XD,ZD)
170 NEXT X
179 REM ** Preserve display **
180 GOTO 180
```

Listing 1. Program to display Z = 10★SIN(X)/X on the VDU.

Fig. 2 A two-dimensional function, Z = 10★SIN(X)/X.

steps of 0.1 there will be 301 values of Z, so we dimension the array. The upper limit of 15.05 in the FOR statement is to compensate for inaccuracies in the single precision maths of the TRS-80. If it starts at −15 and adds 0.1 for 300 times, instead of reaching 15, it gives just over 15 and the loop would finish without calculating that point unless we make the upper limit just over 15.

Line 200 uses ESC A 8 to set the paper advance at a linefeed to 8/72nds inch. As each dot is 1/72nd inch apart this means there will be no spaces between lines. The vertical spacing of dots will be even.

The value of Z will lie between 10 and −3, and for each value we start a new line, set a tab to centre the picture and set the graphics mode.

If the value of Z(N) is less than V then we don't want to plot it yet, so A is set to zero. If Z(N) is not less than V then we have a point that we wish to plot and we must calculate which of the eight dot wires to fire.

As we only wish to plot a single dot, the value to send to the printer will be 1, 2, 4, 8, 16, 32, 64 and 128, which are all powers of 2 ie 2^0, 2^1, ... 2^7. If we subtract the integer part of Z(N) from Z(N) we are left with the remainder RZ, which will be a positive number less than 1. This is multiplied by 8 and the integer taken to give a whole number between 0 and 7 proportional to RZ. This is the exponent EX of 2 we need to print the correct dot. Z(N) is made equal to −100 so it will play no further part.

A peculiarity of the TRS-80 is that it will not LPRINT CHR$(0), (10), (11) or (12); if any of these values might be used we have to POKE the value to the printer which is located at memory byte 14312. The MX-80 puts a 63 on address 14312 when it is ready to accept data which explains line 300. If your computer can LPRINT any number then replace lines 300 and 310 with:

```
300 LPRINT CHR$(A)
```

You can experiment by substituting your own equations in line 140, using an error trap in line 130 if required. You should have a good idea of the upper and lower limits on the value of Z(N) so you can set the loop in line 210.

Alternatively you can have the program detect these for you by adding the lines:

```
105 ZL=50000: ZH=-ZL
143 IF Z(N)<ZL THEN ZL=Z(N)
146 IF Z(N)>ZH THEN ZH=Z(N)
210 FOR V=INT(ZH) TO INT(ZL)
    STEP -1
```

Good equations to try are

```
Z(N) = 3*SIN(X) + 3*COS(2*X)
Z(N) = 1/2*(X/4)^3 - X
```

BUFFERED PRINTING

The second method we will look at is analogous to printing on a memory-mapped VDU screen. We select a block of memory as a buffer in which we will plot all the points to be printed and then dump the buffer out to the printer.

For example, if we wish to plot Z=10★SIN(X)/X as before with X varying from −15 to +15 in steps of 0.1 and Z between −3 and +10, then we need a buffer 301 bytes wide and 14 bytes high: see Fig. 3. So a continuous block of 4212 bytes must be selected and protected. For a TRS-80 with 16K RAM, plenty of

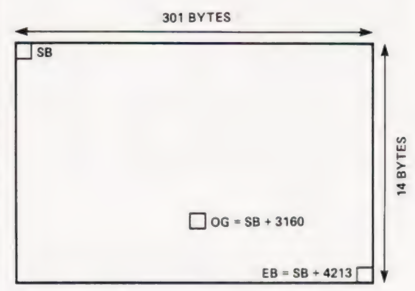


Fig. 3 The print buffer for a two-dimensional function.

memory can be protected by answering the MEM SIZE? question at switch on with 20000. Whatever your system, choose a suitable value for the start byte SB.

SB represents X−15 and Z=10, so the point for X=0, Z=0 is SB+150 (to make X zero) +3010 (to make Z zero. Thus the origin is OG=SB+3160. The end byte EB is SB+4213.

Listing 3 shows the program for plotting the function. Notice the similarities in structure between this program and Listing 1, where the function was plotted on the VDU.

The buffer area must first be cleared to avoid printing random points, then each point is calculated and plotted in the buffer. As before, the appropriate bit is derived by raising 2 to an exponent formed from the remainder of Z.

When plotting the point, the integer part of Z is multiplied by 301, the number of bytes in a line, in order to plot the point in the correct line. Because the top line to be printed is from the lowest memory addresses in the buffer, Z is subtracted from the origin rather than added.

When dumping the buffer, C is used to count the number of bytes printed in a line and when it reaches 301 a linefeed is sent, followed by data to set up the bit image mode for

```
99  REM ** Dimension array **
100 DIM Z(301)
109 REM ** Calculate function **
110 FOR X=-15 TO 15.05 STEP .1
120    N=INT(X*10)+151
130    IF X=0 THEN Z(N)=10: GOTO 150
140    Z(N)=10*SIN(X)/X
150 NEXT X
199 REM ** Set line spacing **
200 LPRINT CHR$(27);CHR$(65);CHR$(8)
210 FOR V=10 TO -3 STEP -1
219    REM ** Prepare graphics mode **
220    LPRINT TAB(15);CHR$(27);CHR$(75);CHR$(45);CHR$(1);
230    FOR N=1 TO 301
240       IF Z(N)<V THEN A=0: GOTO 300
250       RZ=Z(N)-INT(Z(N))
260       EX=INT(8*RZ)
270       A=2[EX
280       Z(N)=-100
299       REM ** Print point **
300       IF PEEK(14312)<>63 THEN 300
310       POKE 14312,A
320    NEXT N
330    LPRINT
340 NEXT V
349 REM ** Reset printer **
350 LPRINT CHR$(27);CHR$(64)
```

Listing 2. Progam to store a function in an array before printing.

```
99  REM ** Select buffer address **
100 SB=20000: OG=SB+3160: EB=SB+4213
109 REM ** Clear the buffer **
110 FOR B=SB TO EB
120    POKE B,0
130 NEXT B
139 REM ** Calculate function **
140 FOR X=-15 TO 15.05 STEP .1
150    IF X=0 THEN Z=10: GOTO 170
160    Z=10*SIN(X)/X
169    REM ** Scale coordinates for plotting **
170    XD=INT(10*X)
180    ZD=INT(Z)
190    RZ=Z-ZD: EX=INT(8*RZ): A=2[EX
199    REM ** Plot point **
200    POKE OG+XD-301*ZD,A
210 NEXT X
219 REM ** Set line spacing **
220 LPRINT CHR$(27);CHR$(65);CHR$(8)
229 REM ** Set counter **
230 C=301
240 FOR B=SB TO EB
250    IF C<>301 THEN 270
260    LPRINT CHR$(13);TAB(15);CHR$(27);CHR$(75);CHR$(45);CHR$(1);: C=0
269    REM ** Print point **
270    IF PEEK(14312)<>63 THEN 270
280    POKE 14312,PEEK(B)
290    C=C+1
300 NEXT B
309 REM ** Reset printer **
310 LPRINT CHR$(27);CHR$(64)
```

Listing 3. Program to store a function in a buffer before printing.

```
99  REM ** Select buffer address **
100 SB=20000: OG=SB+6308: EB=SB+12616
109 REM ** Clear the buffer **
110 FOR B=SB TO EB: POKE B,0: NEXT B
119 REM ** Calculate the function **
130 FOR Y=15 TO -15 STEP -1
140    PRINT Y
160    FOR X=-15 TO 15.05 STEP .1
170       R=SQR(Y*Y+X*X)
180       IF R=0 THEN Z=10: GOTO 210
190       Z=10*SIN(R)/R
199       REM ** Scale coordinates for plotting **
210       ZD=Y+Z: IZ=INT(ZD)
220       RZ=ZD-IZ: EX=INT(8*RZ): A=2[EX
230       XD=INT(10*X): YD=-341*IZ
240       B=OG+XD+YD
249       REM ** Check point is in bounds **
250       IF B>EB THEN GOTO 310
260       IF B<SB THEN B=OG+XD-6138: A=0
269       REM ** Erase hidden lines **
270       FOR M=OG+XD+6138 TO B+341 STEP -341
280          POKE M,0
290       NEXT M
299       REM ** Plot point **
300       POKE B,(PEEK(B) OR A) AND (256-A)
310    NEXT X
330 NEXT Y
399 REM ** Set line spacing and counter **
400 LPRINT CHR$(27);CHR$(65);CHR$(8): C=341
409 REM ** Print out buffer **
410 FOR B=SB TO EB
420    IF C<>341 THEN 440
430    LPRINT CHR$(13);TAB(11);CHR$(27);CHR$(75);CHR$(85);CHR$(1);: C=0
439    REM ** Print point **
440    IF PEEK(14312)<>63 THEN 440
450    POKE 14312,PEEK(B)
460    C=C+1
470 NEXT B
479 REM ** Reset printer **
480 LPRINT CHR$(27);CHR$(64)
```

Listing 4. Program to print a three-dimensional function.

```
99  REM ** Select buffer address **
100 SB=20000: OG=SB+6308: EB=SB+12616
109 REM ** Clear the buffer **
110 FOR B=SB TO EB: POKE B,0: NEXT B
119 REM ** Calculate the function **
120 C=10
130 FOR Y=12 TO -12.05 STEP -.1
140    IF C=10 THEN S=.1: C=0: PRINT Y: GOTO 160
150    S=1
160    FOR X=-12 TO 12.05 STEP S
170       R=SQR(Y*Y+X*X)
180       IF R=0 THEN Z=10: GOTO 200
190       Z=10*SIN(R)/R
199       REM ** Scale coordinates for plotting **
200       X1=(X+Y)*.7071: Y1=(Y-X)*.7071
210       ZD=Y1+Z: IZ=INT(ZD)
220       RZ=ZD-IZ: EX=INT(8*RZ): A=2[EX
230       XD=INT(10*X1): YD=-341*IZ
240       B=OG+XD+YD
249       REM ** Check point is in bounds **
250       IF B>EB THEN GOTO 310
260       IF B<SB THEN B=OG+XD-6138: A=0
269       REM ** Erase hidden lines **
270       FOR M=OG+XD+6138 TO B+341 STEP -341
280          POKE M,0
290       NEXT M
299       REM ** Plot point **
300       POKE B,(PEEK(B) OR A) AND (256-A)
310    NEXT X
320    C=C+1
330 NEXT Y
399 REM ** Set line spacing and counter **
400 LPRINT CHR$(27);CHR$(65);CHR$(8): C=341
409 REM ** Print out buffer **
410 FOR B=SB TO EB
420    IF C<>341 THEN 440
430    LPRINT CHR$(13);TAB(11);CHR$(27);CHR$(75);CHR$(85);CHR$(1);: C=0
439    REM ** Print point **
440    IF PEEK(14312)<>63 THEN 440
450    POKE 14312,PEEK(B)
460    C=C+1
470 NEXT B
479 REM ** Reset printer **
480 LPRINT CHR$(27);CHR$(64)
```

Listing 5. Program to print an oblique view of a three-dimensional function.

the next line: C is then reset to zero. It is set to 301 initially in order to print the first line.

If we compare the two methods we see that the first calculates the values of Z quite quickly, although it is a bit slow at plotting the function. The second method uses more memory which has to be protected, takes time to clear the buffer and calculate each point, but is faster at printing out. Both ways are comparable, but the second method of plotting points in a buffer before dumping it to the printer is more versatile, as we shall see when we try to plot three-dimensional functions.

3D FUNCTIONS

Let's look at the plotting of 3-dimensional functions. In these cases we will plot Z as a function of X and Y, or maybe R where R is the distance from the origin. We will plot values of Y above each other to generate the impression of depth.

We will require a larger buffer for these functions, so we will set one which will be used for the remainder of the article. The buffer will be 341 bytes wide by 37 bytes deep, and on my system occupies memory locations 20000 to 32616: see Fig. 4.

When we come to use oblique views, this window will allow plotting of functions with X and Y both varying from −12 to +12. Once again, select a suitable value of SB for your system.

The simplest way to proceed is to select successive values of Y starting in the background and working towards the foreground. For each value Y, a line is plotted by calculating the function Z. Each point is plotted in the buffer as it is calculated.

By starting at the back we can cater for the case of hidden lines, ie lines which will be covered by plotting the function nearer the front. This is done by calculating where a point is to be plotted and erasing any previously plotted points which lie below it.

We are now working in three dimensions so Z can be a function of X and Y or, if it is circularly symmetrical, a function of R, the distance from the origin. In this case, using Pythagoras's theorem, R=SQR(X★X+Y★Y).

As an advance from the function we plotted in two dimensions, let's look at plotting Z=10★SIN(R)/R. The program to do this is in Listing 4 and the result in Fig. 5.

First the buffer is defined and then cleared. Following this are a pair of nested loops to calculate all the points for each value of Y and X. There are 31 values of Y from −15 to 15 and 301 values of X by going in steps of 0.1. So in total there are 9331 points to plot. This takes about an hour to complete, so to keep track of progress and to satisfy yourself the computer hasn't hung up the PRINT Y statement at line 140 is included.

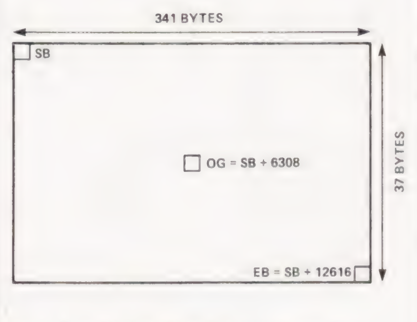


Fig. 4 The print buffer for three-dimensional functions.

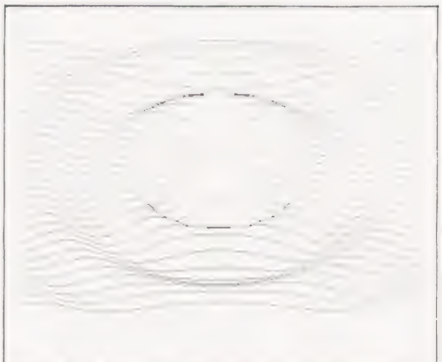


Fig. 5 A three-dimensional function, Z = 10★SIN(R)/R.

This is probably a good place to point out that the listings in this article have been written for ease of understanding and to indicate the structure of the program. Significant savings can be made if spaces and remarks are deleted, multi-statement lines are used and series of calculations are combined, especially within the inner loop.

As an example, line 170 calculates Y★Y 301 times for each value of Y. If this is calculated once as Y2=Y★Y in line 140 then R=SQR(Y2+X★X) can be used in line 170. Another saving could be to convert line 220 to A=2^(INT(8★(ZD−IZ))).

In a manner similar to the two-

dimensional case, the value of Z is calculated, the point to plot is derived and stored in A and the byte in which the point is located is calculated by adding an X and Y displacement to the origin. This takes place in lines 170 to 240.

Lines 250 and 260 then check if the byte is within the bounds of the buffer. If it is after the end byte EB the point is ignored. If it is before SB then the point is not plotted but it will be in front of any points already plotted below it in the buffer. In this case the byte is converted to the top row of the buffer to enable hidden line removal and A is reset to 0 so no point will be plotted.

Hidden lines will be erased by the routine in lines 270 to 290. As we plot the function from the back forward, any previously plotted point will be behind the point currently being plotted. If it is also below the current point then it is 'hidden' and must be erased. The routine starts at the bottom of the buffer and resets to zero all bytes directly below the byte in which the point is to be plotted.

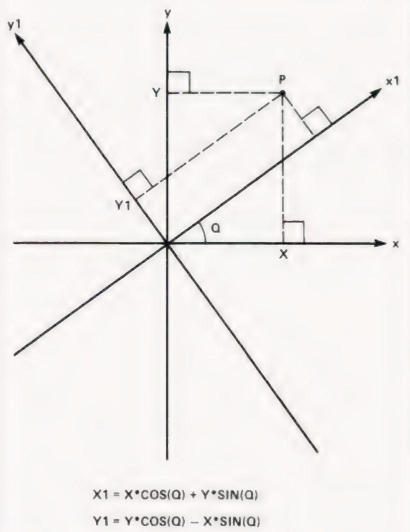


Fig. 6 Relationships for the rotation of axes.

There is still a problem of hidden points within the byte as each bit represents a separately plottable point. More significant bits than the one in which the point is plotted are not hidden and most be retained, but the less significant bits must be erased. The logical expression in line 300 accomplishes this.

A is a power of 2 and represents one bit. If ORed with PEEK(B) it is equivalent to setting that bit. (PEEK(B) OR A) therefore represents the contents of B with the bit set that we are interested in. For example, if B has two bits set, say the LSB and MSB, and A = 16, then:

```
         PEEK(B) = 1 0 0 0 0 0 0 1
               A = 0 0 0 1 0 0 0 0
  (PEEK(B) OR A) = 1 0 0 1 0 0 0 1
```

Fig. 7 An oblique view of the function Z = 10★SIN(R)/R.

Remembering that A is a power of two, a little experimenting will show that (256 – A) represents a binary number with all bits above and including A set, and all below A reset.

```
          A = 16 = 0 0 0 1 0 0 0 0
(256 - A) = 240 = 1 1 1 1 0 0 0 0
```

(256 – A) represents a mask which passes the bits we want and blocks the others.

```
(PEEK(B) OR A) AND (256 - A) =
1 0 0 1 0 0 0 0
```

Once all the points have been plotted, the buffer is dumped to the printer as in the two-dimensional case, except that there are now 341 bytes per line.

Fig. 8 Some other functions.

OBLIQUE VIEWS

Sometimes it is of value to view a contour obliquely and plot the lines parallel to the X and Y axes. This can be readily achieved with small changes to our program.

First of all, consider how to represent a point viewed obliquely. Suppose we have a point (X,Y) that we wish to view from an angle Q (see Fig. 6). The new coordinates (X1,Y1) are calculated by:

Take in CRA 8

For the special case of 45°:

```
COS(Q) = SIN(Q) = 0.7071
X1 = (X + Y)*0.7071
Y1 = (Y - X)*0.7071
```

To produce Fig. 7 this change is included in Listing 5 at line 200, along with other changes and additions. First of all, X and Y are limited to between +12 and −12 giving a maximum value for X1 of:

```
(12 + 12)*0.7071 = 16.97
```

The minimum value is −16.97 and the same holds for Y1. This uses the whole width and height of the buffer.

For all integer values of Y the value of X is incremented in steps of 0.1, which effectively plots a continuous line parallel to the Y axis. At all other Y values, X is incremented by 1 to build up the lines parallel to the X axis. The value of the step is calculated in lines 140 and 150. C is introduced as a counter to determine when the step of 0.1 is required and is initially set to 10.

With these changes made, the program calculates the function and plots it using X1 and Y1. Plotting the points into the buffer and dumping the buffer to the printer are accomplished in the same way as in the previous listing.

FURTHER DEVELOPMENTS

Considering the time it takes the programs in this article to run, it would be a wise move to write machine language routines to clear the buffer and dump the buffer to the printer. However, by far the longest time is taken in calculating all the points and plotting them in the buffer. A routine to save the buffer to tape or disc could save a lot of time in future if it was necessary to print out a function again.

With a little adaptation these routines could be used to plot functions on the video display if your computer has a high resolution graphics capability. This would give the enjoyment of being able to watch the picture being built up rather than just imagining it.

Don Thomasson

SPECTRUM MACHINE CODE

With a hefty piece of machine code to write on the Spectrum, our reviewer looked around for some tools to help him out. This is what he found.

Facing the task of writing a fairly large machine code program for the Spectrum, I began to cast round for the information and software tools which I would need. The results may be of interest to others who are making similar investigations.

Where information on the contents of the Spectrum ROM is concerned, the obvious authoritarian source is **The Complete Spectrum ROM Disassembly** (Logan & O'Hara, Melbourne House). This gives all the interpreter and operating system details, with terse comments, and all the information one could possibly need is there — somewhere.

Being naturally lazy, not to say slightly pressed for time, I hoped to be able to supplement this data with some more direct information on how the ROM routines could be accessed. A set of three books from Interface was suggested, but these proved a disappointment.

Perhaps I should mention at this point that I have been working in Z80 machine code quite happily for some three years now, having graduated through a number of other processors before that. What I wanted was data relating to the Spectrum. For information on the Z80, it would be difficult to improve on **Programming the Z80** (Rodnay Zaks, Syber), which has long been our standard reference on the subject.

The three Interface books were aimed at comparative beginners, even though one had the subtitle 'For Advanced Programmers'. Starting with explanations of binary and hexadecimal notation, they moved quite briskly forward, but were primarily concerned with machine code as a support to BASIC programs, whereas I needed information on the use of ROM routines to support machine code.

There were also some rather fundamental errors. In the volume subtitled 'For Beginners', it was said that addition could only be performed in the A register, a lengthy explanation being given of a way to perform HL = HL + DE via the A register. The instruction ADD HL, DE would have been more appropriate. It was also stated that the carry flag could not be transferred to the A register, which is nonsense.

These may seem to be very minor points, but such mis-statements have a way of spreading themselves. It seemed most unfortunate, to put it mildly, that beginners should be misled in this way.

It was also a pity that some of the listings were printed as facsimiles of rather tatty ZX Printer output. Now that there are several systems for driving respectable printers from the Spectrum this is unnecessary. One long listing in the largest of the three books was based on the output of a printer which with a very limited character matrix, and that was very difficult to read. However, most of the rest of the book used typeset listings that were very clear.

The first two books were both titled **Spectrum Machine Code Made Easy**, Volume 1 (James Walsh) being subtitled 'For Beginners', and Volume 2 (Paul Holmes) being subtitled 'For Advanced Programmers'. Both authors are teenagers, and while their text showed considerable maturity the content perhaps reflected their limited experience.

The third book, titled **Mastering Code on your ZX Spectrum** (Toni Baker), was much more comprehensive, giving some quite substantial programs as illustrations.

These three books were examined in detail, but several more were examined before it became obvious that there was a distinct gap in coverage, with nothing to provide the data needed by the relatively serious programmer. This is rather typical of the personal computer scene. There is plenty of help for novices, up to a point, but thereafter they are left to their own devices. But for that, many more would be able to continue their development to higher levels.

The next need was for software tools. For programs up to around 600 bytes, I will happily code by hand, but with 6000 bytes and a large database in prospect an assembler was obviously desirable.

The first specimen, the Zeus Assembler from Crystal, failed to please. It gave a display in normal Spectrum lettering, lower case, in black on white, and there was little in the way of formatting to make the result readable. For example, labels tended to get buried in a confused mass of text, where they should have been in a column on their own.

The second offering brought relief. The Picturesque Editor/Assembler was in a different class: the display was a pleasant white on blue, with 40 columns neatly divided into appropriate fields. Capital letters of very readable form were the norm, though lower case became available for text between quotes, used for comments and messages, The Editor provided auto-numbering and renumber, making insertions as easy as in BASIC, and the simple command set provided all the functions needed.

There was the slight snag that the cursor had to be moved to the left-hand column before any command

was recognised, but that soon became a matter of habit. There was also the fact that the all code was assembled in the Object Buffer, starting 256 bytes above STKEND, whatever the stated origin for the code might be. It was then necessary to save the Object Buffer to tape, producing a result that could be loaded into the correct position by the usual LOAD"" CODE command.

Nevertheless, it was possible to test code before saving it by specifying ORG#, which put the origin at the start of the Object Buffer. Once the code was proved, it could be re-assembled with the proper origin, and saved to tape.

Verify was intelligent enough to know what kind of recording was involved, either source text or object code, and it was possible to retain the contents of the Lable Table while loading fresh source code.

In short, the system was a near-professional standard in concept, and thoroughly professional in execution, which is more than could be said of some of its competitors.

The program also has the benefit of a good backup service. A mild query brought a full page letter of explanation, together with a copy of the most recent issue of the tape. In response to a question about driving more respectable printers, the letter also enclosed a listing for a driver to match the Kempston interface, and that was the clincher, since it would allow the programs produced to be properly documented on an Epson MX80.

As a companion to the Assembler, Picturesque produce a Monitor, both programs being supplied in 16K and 48K form. It is therefore possible to load both at the same time, which can save a lot of fiddling about, at the expense of a reduction in working space.

The Assembler is so quick, especially with no display or printout, that there was little excuse for patching code directly, but the Monitor had a number of other uses. It would move code bodily from one area to another, or fill an area of RAM with a given byte value. It would insert or delete code, moving higher code appropriately. It would jump to a specified start point. It would set, implement, and clear breakpoints, display register contents and alter them, generate a hex dump, or enter text into memory. There was also a conversion routine between hex and decimal. The MX80 printer driver will work with the Monitor, too.

Equipped with these two Picturesque programs, I feel able to approach the task ahead with greater confidence. When such satisfactory offerings are available, it seems a pity that many users, buying blind by post, may well think that less professional programs are the best available. Being a cynic, I sometimes suspect that products that are advertised strenuously are less satisfactory than those which are not advertised at all. Getting in touch with Picturesque was a little difficult, because they advertise very little . . .

The Complete Spectrum ROM Disassembly, Dr. Ian Logan and Dr. Frank O'Hara, Melbourne House, £9.95.
Programming the Z80, Rodnay Zaks, Sybex, £9.95 from The Computer Bookshop, 30 Lincoln Road, Olton, Birmingham B27 6PA.
Spectrum Machine Code Made Easy Vol 1, James Walsh, Interface, £5.95.
Spectrum Machine Code Made Easy Vol 2, Paul Holmes, Interface, £5.95.
Mastering Machine Code for your Spectrum, Toni Baker, Interface, £9.95.
Spectrum Editor/Assembler, £8.50 including VAT and postage.
Spectrum Monitor, £7.50 including VAT and postage.
Both the above from Picturesque, 6 Corkscrew Hill, West Wickham, Kent BR4 9BB.

Not all home computers stay at home.

The BBC Micro is the ideal family computer–simple to operate, yet fast, powerful, with enormous potential.

But it's nice to know, when you buy one for your home, that the business, educational and scientific worlds agree with your choice.

Here are a few stories to illustrate how the BBC Micro gets out and about. And one to remind you how helpful it can be when it stays at home.

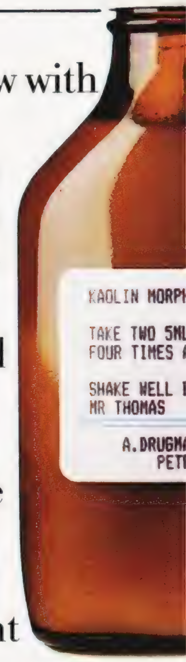
A practical lesson in business admin.

The contribution of the BBC Micro in the classroom has long been recognised at Perins Community School in Hampshire.

The School has 12 BBC Micros used extensively across the syllabus: in fact some pupils are using them to study for their GCE O Levels in computing.

One of the programs available to Perins teachers such as David Beck, pictured below with his class, is "Newsagent."

This program contains all the necessary information for the class to run a newsagent's shop; allowing them to organise daily deliveries, make up bills and keep an eye on stock control and ordering.



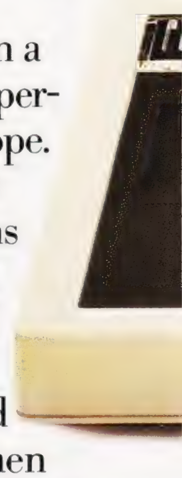
It's a nice example of how the BBC Micro can be used not only to acquaint a class with the language of computers, but also with some of the realities of the community in which they live.

Correcting Jodrell Bank.

The BBC Micro is a familiar worker around Jodrell Bank.

You'll find it in the reception area explaining the workings of a radio telescope to visitors, for example.

But it's also been helping in a more testing task: to improve the performance of the Defford telescope.

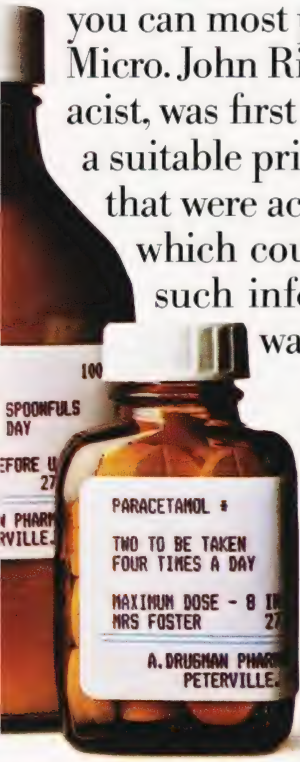


In this application it has been used to make calculations necessary to determine the precise parabolic shape of the dish.

Theodolites are used to do the measuring–then the BBC Micro works out the necessary corrections.

The end of the scrawl.

If any of you have noticed how much easier it is to read and understand labels on drugs and medicines these days, then you can most probably thank the BBC Micro. John Richardson, a Preston pharmacist, was first to realise how a micro with a suitable printer could produce labels that were accurate and legible and which could include, automatically, such information as drug reaction warnings.



At the same time it could record drug usage for better stock control.

He chose the BBC Micro for its versatility and potential for expansion.

John Richardson believes that this system will be recognised as standard in the profession and be used in hospitals, health centres and pharmacies throughout the UK.

Meanwhile back at home.

Dr. & Mrs. Yarwood bought a BBC Micro as a birthday present for their 12 year old daughter.

However, it quite quickly became common property.

All three can now write their own programs. Mrs. Yarwood is particularly proud of one program she has compiled to help teach her daughter French vocabulary.

They all agree that although the Micro is fast and powerful enough to be at home in Jodrell Bank, it is also the ideal computer at the Yarwood home: simple to set up (virtually any TV set and cassette player is all you need) and simple to use.

All this for only £399.

The BBC Micro comes with a comprehensive, step-by-step User Guide which introduces you to your micro and shows you how to construct useful programs of your own.

You will also receive a free "Welcome" cassette which contains 15 different programs for you to experiment with, ranging from music and graphics to games like Kingdom and Bat 'n' Ball.

The BBC Micro is available from W H Smith Computer Shops, Boots, John Lewis and local Acorn stockists.

Alternatively if you would like to order one with your credit card or if you want the address of your nearest supplier just phone 01-200 0200 or 0933-79300.

The BBC Microcomputer System.

Designed, produced and distributed by Acorn Computers Limited.

Tony Cross and Phil Cornes

GETTING MORE FROM THE 64 PART 3

Our final article in this series looks at the sound capabilities of the Commodore 64 (and some of the errors in the User Manual!)

There is no doubt that good sound effects can make even the simplest of games much more exciting to play. But the use of sound in computing need not be restricted to the games applications. Sound, when used properly, can provide as much information as several lines of text. Warning tones, advisory tones, audio feedback on data entry and, on the more advanced machines, real music are all possibilities.

The sound system on the Commodore 64 is well able to provide almost limitless sound effects and excellent music. And because the sound facilities are accessed by POKEing individual registers the system is very flexible. This allows experimentation with all sorts of unlikely combinations of tones and filters when developing sound effects.

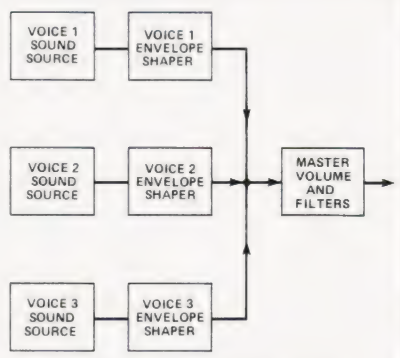


Fig. 1 The organisation of the voices.

THE SOUND SYSTEM ORGANISATION

The sound system is controlled by the Sound Interface Device or SID chip for short. Internally it has three completely separate sound channels or voices. The voices are organised as shown in Fig. 1.

Each of the sound sources can produce four different waveforms, triangular, sawtooth, rectangular (pulse) and white noise. The frequencies of each of the waveforms can be individually varied and the pulse width ratio of the rectangular waveform can also be varied. Figure 2 shows the shapes of the four different waveforms. For any particular note, these four waveforms produce very different sounds.

The sound is then fed into an envelope shaper where the Sustain level and the Attack, Decay and Release rates can be set up (the ADSR envelope). Figure 3 shows a typical ADSR envelope. It is this envelope shape which gives musical instruments their distinctive sounds and it is possible to 'simulate' some of them quite well. Alternatively new 'computer instrument' sounds can be produced by using other envelope shapes.

Finally the outputs from the three voices are brought together under a master volume control. Three types of tone filtering can also be introduced at this stage. These are a high pass, a low pass and a band pass filter. The cutoff frequencies of these filters can be individually selected. Figure 4 shows the effects of the three types of filter.

THE SOUND SOURCES

Commodore call the sound sources 'waveform generators' and they have two main functions:

- To produce a note of the selected frequency in the selected waveform.
- To enable the waveform generator output to be turned on or off.

For each voice both these functions are controlled by the same register, the voice control register. These control registers are at the following locations:

Voice 1 = 54276
Voice 2 = 54283*
Voice 3 = 54290

*NOTE — this register is incorrectly specified as 54288 in our version of the User Manual.

The contents of each of the control registers are very similar (we will look at the differences between them later). The bits within the registers are used as follows:

7 White noise select (1 = ON)
6 Pulse waveform select (1 = ON)
5 Sawtooth waveform select (1 = ON)
4 Triangle waveform select (1 = ON)

Bits 1-3 do not concern us here. Leave them at 0.

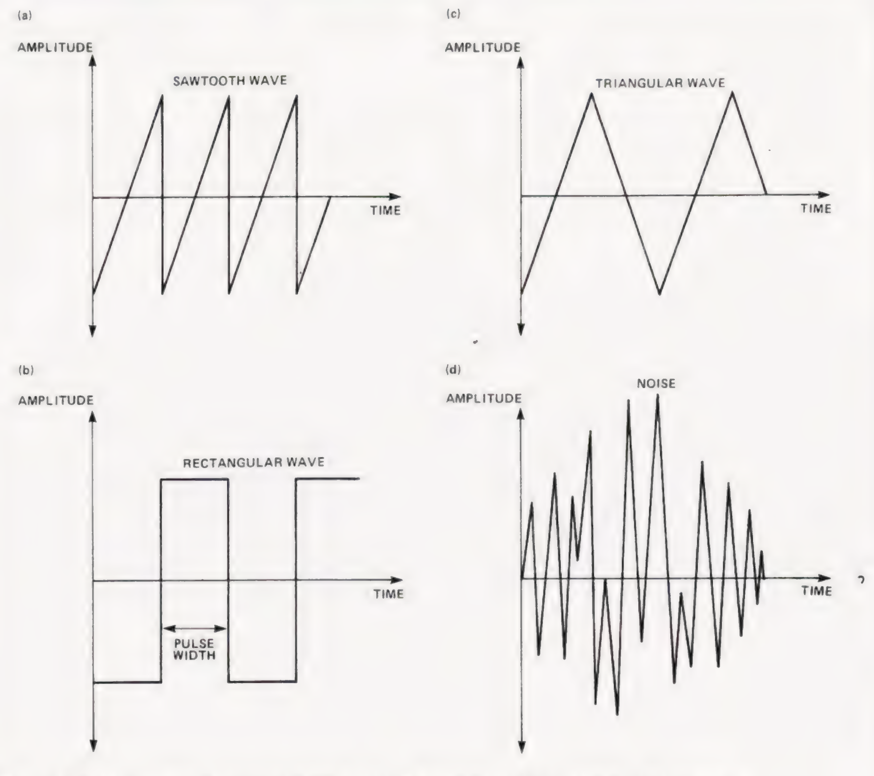


Fig. 2 Four different waveforms. (a) Sawtooth (b) Triangular wave (c) Rectangular wave (d) Noise.

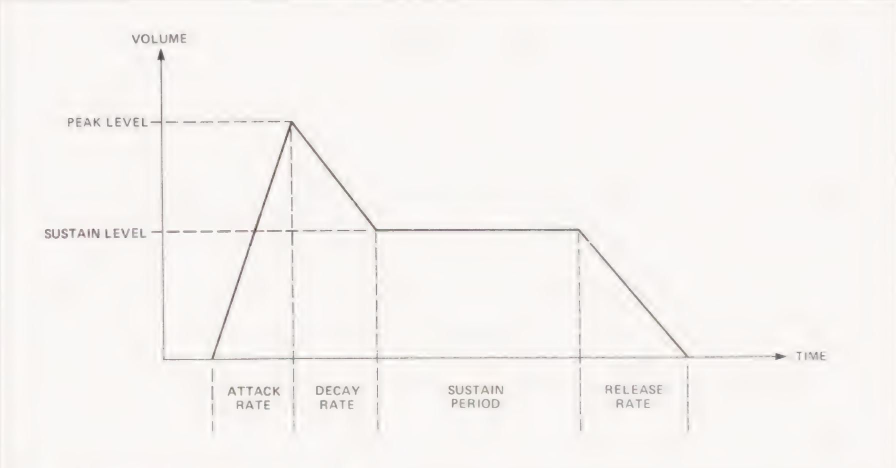


Fig. 3 A typical ADSR envelope.

0 Gate bit (1 = Start attack/decay/sustain cycle).
(0 = Start release cycle)

From this you can see that there are two decimal values for each waveform type: one for the attack/decay/sustain cycle and one for the release cycle. Table 1 shows the two values of each waveform.

The procedure for playing any given note is:

- Start attack/decay/sustain cycle (POKE voice control register, Attack/decay/sustain value).
- Hold sustain for length of note (Count note length).
- When note has finished, start release (POKE voice control register, Release value).

The length of the note depends on the type of note being played (crotchets, quavers and so on all have different lengths). The obvious way to time this delay is with a simple FOR-NEXT loop. You will need to determine by trial and error the delay required for a crotchet to sound the right length, but having found this value all the other note lengths are simply multiples of this value.

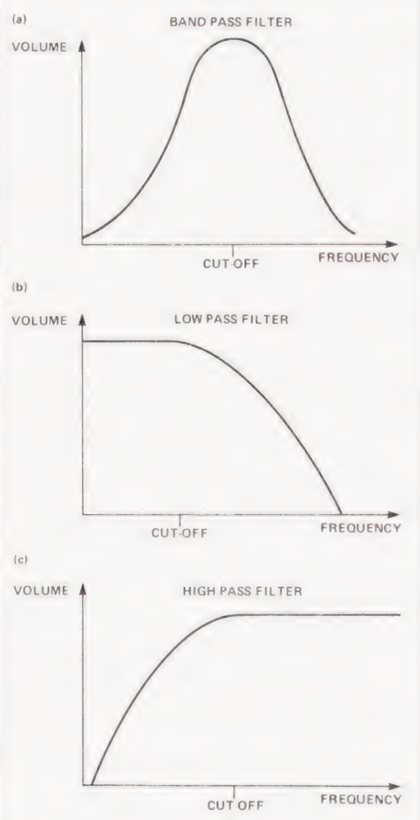


Fig. 4 The effects of various types of filter. (a) Band pass (b) Low pass (c) High pass.

PITCHING IT RIGHT

Selecting the frequency of the desired note is a little bit more difficult. For each voice the frequency of the waveform generator output is determined by the values in two registers, the high frequency control register and the low frequency control register. Table 2 shows where the frequency control registers for each voice are located.

The two frequency control registers are 'read' by the SID chip as a single 16-bit register giving 65536 different selectable frequency 'steps'. The waveform generators in the SID chip can generate frequencies in the range 0 to 4000 Hz, so that each selectable frequency step changes the frequency by only 0.06 Hz. Now, we don't believe that anyone can tell the difference between two adjacent steps that are only 0.06 Hz apart!

Within this range of frequencies is a full eight octaves of musical notes. Appendix M in the User Manual lists the high and low frequency values needed to produce 95 of the 96 notes. (More about the 96th note later). These values should not be taken as the gospel truth — indeed a musician would throw up his hands in horror if he saw them! The reason is to do with the way in which the musical scale is constructed. The root note of the scale is usually taken to be A above middle C, which is in octave 4 of appendix M. This note has a frequency of exactly 440 Hz at concert pitch and all computers and synthesizers calculate the other notes from this one such that each note is 2^1/12 (the twelfth root of two) times the previous note. 'Real' notes are not exactly this distance apart but you need a good ear to be able to tell the difference. Table 3 gives a comparison between the Commodore 64 frequencies and 'real note' frequencies for A above middle C and middle C.

The result of all this is that if the values listed in appendix M do not sound right to you then feel free to change them quite drastically. (At 0.06 Hz a step some values may change by quite a lot!)

Finally, when using the rectangular waveform, there is a little bit more work to be done because we also have to set up the pulse width ratio. This value defines how wide the high part of the rectangular wave will be. (See also Fig. 2). Different

Waveform type	Attack/decay/sustain Gate bit = 1	Release Gate bit = 0
Noise	129	128
Pulse	65	64
Sawtooth	33	32
Triangle	17	16

Table 1. Control register values for the various types of waveform.

Voice number	High Pulse Width register	Low Pulse Width register
1	54275	54274
2	54282	54281
3	54289	54288

Table 2. Locations in memory of the frequency control registers.

Note name	Commodore frequency	Real note frequency
A above middle C	440.0 Hz	440.0 Hz
middle C	261.6 Hz	264.0 Hz

Table 3. Frequency comparison table.

values of pulse width can produce an enormous difference in the sound of rectangular waves.

For each voice the pulse width is specified as a 12-bit number, giving 4096 different steps. The high four bits of this number are specified in the High Pulse Width register and the low eight bits are specified in the Low Pulse Width register. Table 4 shows the locations of the high and low pulse width registers for each voice.

The system has been arranged so that a pulse width of 4095 gives an output that is permanently high (maximum width) and a pulse width of 0 is permanently low (minimum width). In both cases the output will be zero. Normally, of course, we will be using values in between these two — typically 2048 which produces a pure square wave.

Voice number	High frequency control register	Low frequency control register
1	54273	54272
2	54280	54279
3	54287	54286

Table 4. Locations in memory of the high and low pulse width registers.

Note name	High frequency	Low frequency	16-bit frequency
C	137	43	35115
C#	145	83	37203
D	153	247	39415
D#	163	31	41759
E	172	210	44242
F	183	25	46873
F#	193	252	49660
G	205	133	52613
G#	217	189	55741
A	230	176	59056
A#	244	103	62567
B	258	241	66289*

* NOTE — this value is not included in the list in appendix M because it is greater than 65535 and therefore it cannot be 'played' by the SID chip. We have included it for completeness, partly because it is the 96th note and completes the eight octaves, but mainly so that the 'B' note of the lower octaves can be calculated from it.

Table 5. Notes in the highest octave and their 16-bit frequencies.

SCALING THINGS UP

When it comes to actually storing the different notes you clearly don't want to have to store the values for them all, so some means of calculating them is required. The simplest way is to make use of the fact that the notes in one octave are twice the frequency of the notes in the octave below. This means that if we store the high and low frequency values for the highest octave we can calculate all the other octaves by successively dividing by 2.

To do this easily the high and low frequency values must be combined into one 16-bit value. This can be done as follows:-

```
16-bit freq = (high freq)*256 +
              low freq
```

Incidentally, the inverse of this which restores the high and low values from a 16-bit value is:

```
high freq = INT(16-bit freq/256)
low freq = 16-bit freq - high freq
```

If you were to calculate the 16-bit value of the highest octave notes in appendix M of the User Manual you would obtain the values given in Table 5.

The procedure for calculating any note is:

Get the name and octave of the required note.
Read the 16-bit frequency value for the given note from the highest octave values given in Table 4.
IF the octave number is 7
THEN
  Do nothing
  ELSE
  FOR count = 6 TO octave number STEP −1
    Frequency value
    = frequency value / 2
  NEXT count

Recover high and low frequency values from the 16-bit frequency value remaining and POKE these values into the high and low frequency control registers.

SHAPING UP

So far we have obtained a particular note being 'played' in a particular waveform. To make the note sound as though it were coming from, say, a violin or a guitar, we need to adjust its attack, decay and release rates and its sustain level. The effect that each of these parameters has on a note can be seen by referring to Fig. 3.

The Attack rate: This is the rate at which the note rises from zero to peak volume when the gate bit is set to 1.

The Decay rate: This is the rate at which the note falls from its peak volume to its 'average' or sustain level.

The Sustain level: This is the proportion of the peak volume that the decay rate will fall to.

The Release rate: This is the rate at which the note dies away when the gate bit is set to 0, although very often a new note starts before the old note has fully died away.

Each of these parameters can be varied in 16 different steps and since 16 states can be coded into four bits of binary, two parameters can be fitted into one eight-bit register. For each voice, the attack and decay parameters are contained in the Attack/Decay cycle control register and the sustain and release parameters are contained in the Sustain/Release cycle control register. Table 6 shows where the two cycle control registers for each voice are located.

Within the Attack/Decay cycle control registers, the high four bits define the attack rate and the low four bits define the decay rate. Table 7 shows the rates which the different values of attack and decay can select. Within the Sustain/Release cycle control registers the high four bits define the sustain level and the low four bits define the release rate. Table 7 also shows the rates which the different release values can select, and the levels which the different sustain values can select.

Appendix P of the User's Manual gives some sample ADSR settings for 'simulating' some common musical instruments. Personally we're not very impressed by the results, because there's a lot more 'real' sound than just the waveform and envelope shape. We think that it is much more satisfying to create new and different sounds. (After all you could have bought a real instrument for the price you paid for the Commodore 64!).

USING THE FILTERS

You don't have to use the filters at all if you don't need them. In fact, you can produce hours of perfectly acceptable music without ever even thinking about them. The filters are really for 'fine tuning' a sound, to get just the quality you desire. For this reason they probably have more uses in sound effects than in music.

In any case, the sort of filtering you want to do will depend on the waveform you are using. This is because the different waveforms all

Voice number	Attack/Decay cycle control register	Sustain/Release cycle control register
1	54277	54278
2	54284	54285*
3	54291	54292

* NOTE — this register is incorrectly specified at 54286 in our version of the User Manual.

Table 6. Locations in memory of the cycle control registers.

Register value	Attack rate	Decay/Release rate	Sustain level
0	2 mS	6 mS	0 %
1	8 mS	24 mS	6 %
2	16 mS	48 mS	13%
3	24 mS	72 mS	20 %
4	38 mS	114 mS	26 %
5	56 mS	168 mS	33 %
6	68 mS	204 mS	40 %
7	80 mS	240 mS	46 %
8	100 mS	300 mS	53 %
9	250 mS	750 mS	60 %
10	500 mS	1.5 S	66%
11	800 mS	2.4 S	73 %
12	1 S	3 S	80 %
13	3 S	9 S	86 %
14	5 S	15 S	93%
15	8 S	24 S	100 %

Table 7. Control register values and their effects.

have different harmonic structures.

Harmonics are waves produced in addition to the generated or 'fundamental' wave. They are always integer multiples of the fundamental frequency and they have names which reflect this. For example, the second harmonic has a frequency of twice the fundamental, the third harmonic a frequency of three times the fundamental, and so on.

Some waveforms, like the sawtooth waveform, contain all the harmonics, others, like the triangular waveform, contain only the odd harmonics. In addition, the amount of each harmonic present depends upon the waveform. The triangular waveform, for example, contains harmonics in proportion to the reciprocal of the square of the harmonic number: others, like the sawtooth waveform, contain harmonics in proportion to the reciprocal of the harmonic number alone.

When you decide to use the filters it is mainly the harmonics which you will be filtering out because, like the envelope shape, they contain a lot of the 'individuality' of a given sound.

The three filters are actually derived from one programable filter withn the SID chip. This filter can be programmed to act on any combination of the three voices using any combination of the three filter types (high pass, low pass and band pass). The programming is done by using two filter control registers. One, called the Voice Input control register, at location 54295 decimal, selects the voices to be filtered. Table 8 shows which bits select which voices.

The other control register, the Filter Mode control register, at location 54296 decimal, is also the master volume control. Table 9 shows the utilisation of bits within this register.

Having selected the voice and filter combinations we require, all that remains is to set the cut-off frequency for the filter. The cut-off frequency is the frequency at which the filter operates. For example, at the cut-off frequency the low pass filter will start to reject frequencies. As the frequencies increase away from the cut-off the rejection increases. (See also Fig. 4).

The cut-off frequency is specified as an 11-bit number using two registers in a rather strange way. The high eight bits of the cut-off frequency are specified in the high cut-off frequency register at location 54294 decimal. The low three bits are specified in bits 0-2 of the low cut-off frequency register at location 54293 decimal. Bits 3-7 are not used. This gives a total of 2048 different cut-off frequency steps. The SID chip filter can operate over the range 30 to 12000 Hz so that each cut-off frequency step changes the cut-off frequency by about 6 Hz. This should be fine enough control for most applications.

The procedure for using the filters is:
Select the filter type and master volume to be used (POKE 54296, filter type and volume).
Select the cut-off frequency (POKE 54294, high cut-off : POKE 54293, low cut-off).
Select the voices to be filtered (POKE 542295, voice combination).

AND FINALLY

That's all for this month and for this short series on the Commodore 64. We hope that it's been both interesting and useful. And keep watching this space — we'll be back with more Commodore goodies. (Editor permitting of course!).

Bit 0 — Voice 1 filter control (1=Filter, 0=Don't filter)

Bit 1 — Voice 2 filter control (1=Filter, 0=Don't filter)

Bit 2 — Voice 3 filter control (1=Filter, 0=Don't filter)

Bits 3 to 7 — Do not concern us here.

Table 8. Bit functions in the filter voice input control register (located at 54295 decimal).

Bits 0 to 3 — Set master volume (0=Off, 15=Full volume)

Bit 4 — Select low pass filter (1=On)

Bit 5 — Select band pass filter (1=On)

Bit 6 — Select high pass filter (1=On)

Bit 7 — Does not concern us here

Table 9. Bit functions in the filter mode control register (located at 54296 decimal).

CT STANDARDS

Our regular page explaining the meaning of the various symbols we use to make programs portable.

It has been very encouraging to see the number of programs submitted using our standard codes for graphics and other non-printable characters. However, it has also become increasingly clear that some of our readers haven't heard of them and this page is intended to set them out once again.

All standards tend to be irksome to adhere to but the ones laid out here are fairly simple and tend to make software easier to maintain by the programmer and simpler to understand for others.

CONTROL THAT CURSOR

Our original standards have now grown with the times. Machines such as the Commodore VIC which have a dual Shift capability can now be incorporated, as can those systems which use Control key functions.

The recently introduced BBC system offers pre-programmed function keys which, we are glad to say, can also be handled by our original coding system. It's nice to see just how well adapted the original standards have become over the last two years! (Indeed, a whole series of books is using them as its *de-facto* standard.) The standards for the cursor controls are given in Fig. 1.

[CLS]	CLear Screen
[HOM]	HOMe cursor
[CL]	Cursor Left
[CR]	Cursor Right
[CU]	Cursor Up
[CD]	Cursor Down
[REV]	REVerse video on
[OFF]	Turn it OFF
[SPC]	SPaCe
[CTL]	ConTroL key
[fn]	Function key (BBC)
[G<]	Graphic left (VIC/MZ-80A)
[G>]	Graphic right (VIC/MZ-80A)

Fig. 1. Our extended set of cursor control standards includes four new functions.

To indicate more than one of the above, an optional number can be placed within the brackets; [4 CL], etc.

The use of square brackets has raised one or two queries. The reason for this choice is that *most* of the common microcomputer BASICs don't use them for specific functions. In fact, at least one machine provides an added bonus by returning a Syntax Error if they are found, a useful check in case you type them in by mistake.

The code [SPC] was added to the list of cursor control codes to get over the problem of indicating just how many spaces are contained in the gap in the printout. The other common variant of the code for spaces is used by the ZX people. Their choice was ⍽ and this crops up in the various newsletters they publish.

The code [RVS] has caused a few headaches. This is really specific to the PET where the character set can be displayed in reversed video. On machines which don't have this facility you should either find a character in the set which is the reversed image of the one you want and use that or simply ignore it and use anything else you fancy! Don't forget, you may have to look up and alter the values used elsewhere in the program.

THE GRAPHIC SOLUTION

It soon became obvious that the techniques applied to the confusing cursor controls could also be applied to the graphics symbols. The following standard is now in general use in programs published in *Computing Today*.

If a graphics character or characters are to be displayed in a listing (as opposed to POKE codes or CHR$( ) codes) then they are indicated by the method shown in Fig 2.

Several people have asked what the relationship between the POKE value for a character and that of its shifted graphic might be. In general the shifted version of any character will be 64 greater than the value of that character. This applies to both PET and MZ-80K systems in all cases.

This can be taken further to include machines which use a pixel graphics set rather than pre-programmed PET-style characters and the series of codes for these is given in Fig. 3. As is nearly always the case there is one machine to which the standard shown in Fig. 3 does not apply — Tangerine's Microtan/Micron. This machine uses a four by two cell structure for its pixel graphics instead of the Prestel/Teletext three by two cell. The method for calculating the value to assign to 'P' is shown in Fig. 4, and is fortunately nice and simple.

MAKING REMARKS

Many people scorn the use of REMs within programs but, during the development at least, they are extremely useful. One of the documentation methods that we use is to keep our back-up copy of our programs on a 300 Baud CUTS tape with all the REMs in place: the working copy, be it on tape or disc, is REMless in order to save space.

It is also good programming 'manners' to give your REMs odd line numbers:

```
3999 REM ** CRASH PROOF INPUT
4000 INPUT "THE NUMBER OF ENTRIES"; A
```

A remarkable number of submitted programs have jumps that go not to the relevant point in the program, but to the REM statement. This can cause severe problems when re-numbering after removing the REMs.

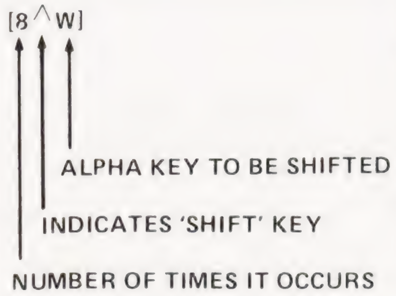


Fig. 2. The way we indicate block graphics on machines like the PET and Sharp. The VIC system of Shift Left and Shift Right is shown in Fig. 1.

1	2
4	8
16	32
64	128

Fig. 4. To convert a Tangerine pixel code into its blocks, simply decode the number into its binary or Hex value and fill in the relevant squares.

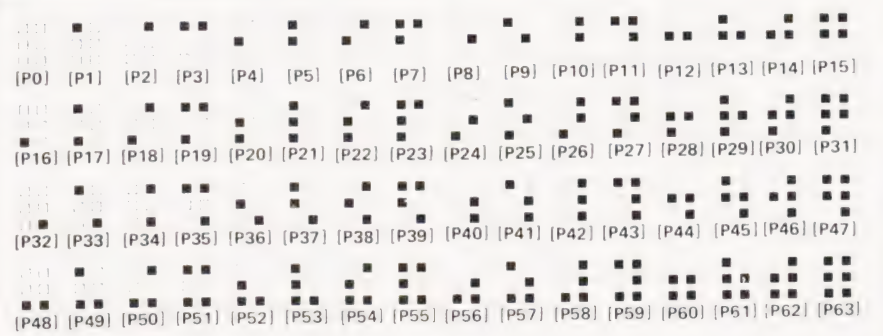


Fig. 3. The standard pixel codes; they will work on most computers which employ this technique as well as for Teletext and Prestel.

Garry Marshall

BOOK PAGE

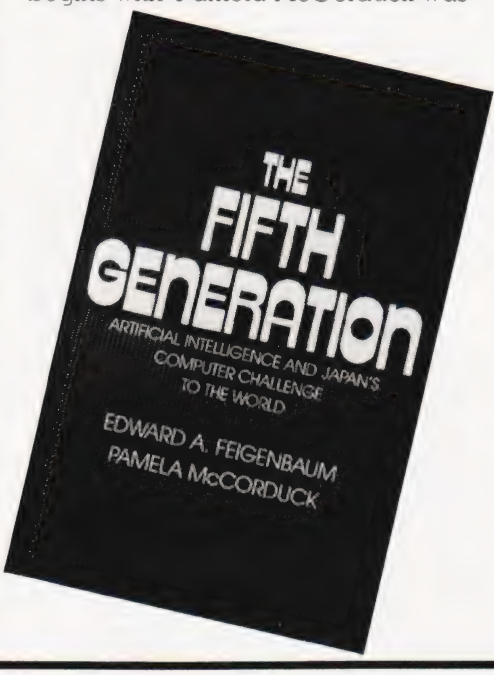
This month our reviewer looks at several books which cover aspects of computing that go beyond simple BASIC programming on your micro.

I reviewed a book on fifth generation computers recently. After spending ages wading through it I wasn't impressed. If only I had waited a bit longer I could have saved myself a lot of time, for this month a book has arrived that gives a splendid explanation of what the fifth generation is all about. It is also probably the most important computer book of the year. Besides this, this month's selection of books includes three books for the BBC Micro that are essentially courses on how to go beyond BASIC with it by using other languages, and one about writing games in Pascal. The sixth and final book brings us neatly in a full circle, because it is about writing expert systems to run on a micro. Expert systems, of course, are at the heart of the fifth generation developments.

The Fifth Generation by Edward A. Feigenbaum and Pamela McCorduck is fundamentally a propaganda job to alert American government and industry to the authors' view that America is losing its lead in computing over Japan at the rate of one day per day as a result of the Japanese national plans for the development of fifth generation computers. This assertion is never really shown to be true, though, and while I don't want to dwell on the politics of fifth generation developments, a couple of remarks may be worth making. There is a fundamental difference between the approaches of the Americans and the Japanese computer companies to research. The American companies are all in intense competition and unlikely to share their findings, while the Japanese companies are collaborating with each other. At the same time a great deal of research is going on in American computer laboratories on the topics that are involved in developing the fifth generation. It has not yet been shown that the Japanese approach is superior to the American. Secondly, the Japanese plans for the fifth generation require that technological breakthroughs be achieved as scheduled activities. This shows clearly the nature of the plans and the degree of risk that is involved. The argument goes that even if the Japanese only achieve a small proportion of their aims they will still achieve an awful lot. But how clear is it that this means they will achieve more than the Americans?

Anyway, to the book. Feigenbaum is Professor of Computer Science at Stanford University and, as head of the Heuristics Programming Project there, is one of the founding fathers of Artificial Intelligence and has established one of the world's major Artificial Intelligence groups. This makes him as well qualified as anybody to assess the fifth generation plans. The first part of the book contains a very readable description of the aims of the fifth generation, of which I shall say no more having devoted a good deal of space to it only recently. In passing, this part of the book demonstrates quite clearly why today's children need to know about computers and computing by showing the extend to which their futures are likely to depend on them. So if anyone still doubts the value of having computers in every school in the country, let them read this book carefully.

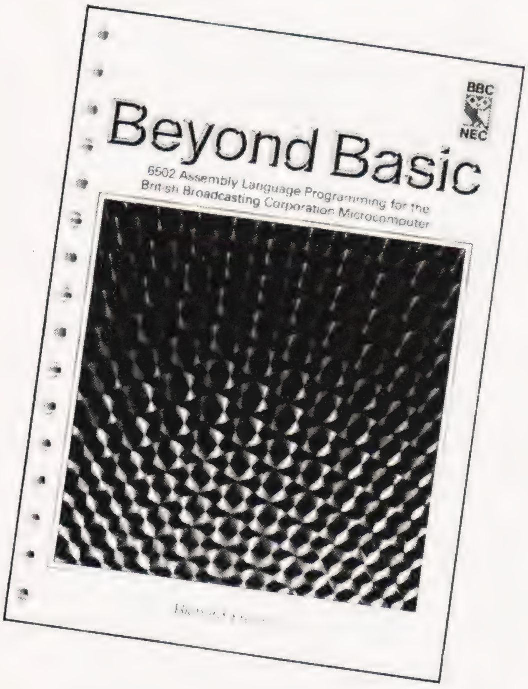
The second author, Pamela McCorduck is a writer on science and I take it to be her influence that has resulted in the most unfortunate style with which some parts of the book are written. To illustrate, Part 2 begins with 'Pamela McCorduck was introduced to the idea of artificial intelligence . . . in 1959, by Feigenbaum as it happened, when computing, its natural child called artificial intelligence and certainly they themselves were all much younger'. Although it is a matter of taste, this makes me cringe: it smacks of 'faction' and tends to diminish the credibility of the book. An account of the development of artificial intelligence proceeds in this style and while Feigenbaum obviously belongs in it (self-referentially or not) it is not clear to me what McCorduck is doing there even if she was a spectator.

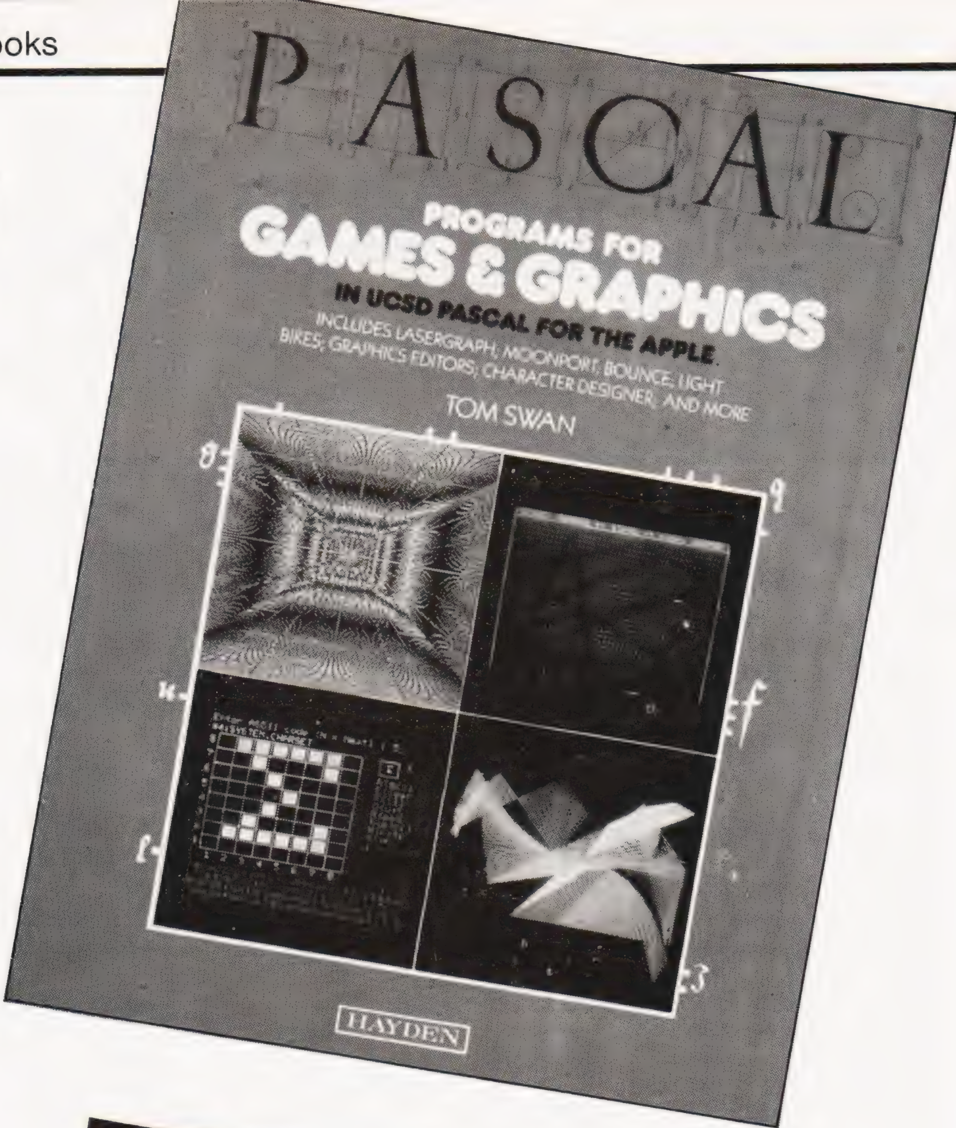


However, if the style of the book is irritating at times, its content is sufficiently fascinating to make up for that. It explains very well that knowledge and information have a value of their own, and unless it is understood that knowledge **is** valuable the whole basis of the fifth generation is inscrutable. That readily available computing power and capability can do so much more than simply store knowledge, and that it can enhance and even create it, is perfectly illustrated by the story of how one of the most influential books on the design of VLSI circuitry, **Introduction to VLSI Systems** by Mead and Conway, was brought to fruition. Its final form, and the short time taken to reach it, both resulted directly from the instant availability of the draft manuscript to many interested parties via the ARPA network. Making the draft manuscript available by this computer communications network allowed many people working in VLSI design to read, test, comment upon and suggest amendments and further material for it. This form of collaboration and opportunity for interaction permitted the book to take a final form, on a very short time scale, which could have been achieved in no other way. Thus, the existence of a computer communications network led directly to the creation of an important body of knowledge.

The book itself is a fine account of all the factors involved in the fifth generation developments. Even with a rather hectoring propagandist approach and its unfortunate style, its contents are never less than fascinating and stimulating.

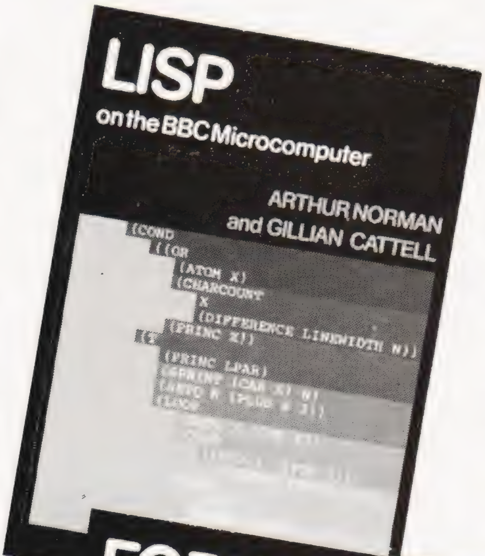
The sub-title of **Beyond Basic** by Richard Freeman is '6502 assembly language programming for the BBC Microcomputer', and this says exactly waht the book is all about. It is a carefully paced course of assembly code programming using the BBC Micro's 'BASIC assembler', with which assembly code segments can be inserted directly into a BASIC program by enclosing them in square brackets. The course covers

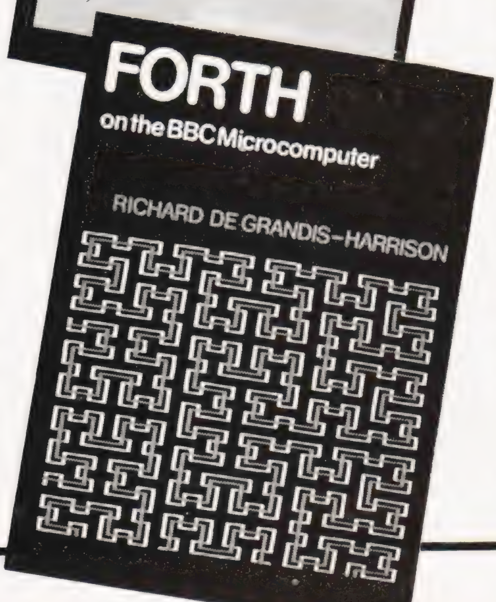




the usual standard material on assembly code programming from simple arithmetic through loops and branches, addressing modes, multiplication and division to the use of the stack, masking and subroutines. It also covers the oddities of the BBC Micro's assembly code programming system, such as how to generate the two-pass assembly process that is necessary, for example, to deal with forward references to labels in assembly code programs. Altogether, the book provides an absolutely standard and solid, though unimaginative, introduction to its subject. It should teach the newcomer to assembly code programming on the BBC Micro everything that is necessary to achieve competence. It is, however, difficult to recommend the book to anyone outside its target audience, as the material on assembly code is covered by many other books in exactly the same form, and the material on the oddities of the BBC Micro is obviously of no interest if you use another computer.

LISP on the BBC Microcomputer by Arthur Norman and Gillian Catell and **FORTH on the BBC Microcomputer** by Richard de Grandis-Harrison are both manuals to accompany Acornsoft's implementations of the respective languages for the BBC Micro. The LISP book is impressive and interesting, but it is my impression that it would be difficult for the newcomer to LISP to progress to the writing of substantial and useful LISP programs with it alone. It presents many examples, but does not spell out the underlying principles of LISP. It also presents a good deal of material that is of no interest to the newcomer to LISP, but which is only of historical interest. It also passes rather quickly from introducing the features of LISP to presenting rather sketchy outlines of medium-sized projects for programming in LISP. Some of the latter are rather imaginative, from a route-finder to an adventure game. Thus, although not well suited to the beginner, it is valuable as a source of reference and as a source of ideas to the fairly experienced LISP programmer with a BBC Micro. This, however, makes the area of interest in the book a rather restricted one.

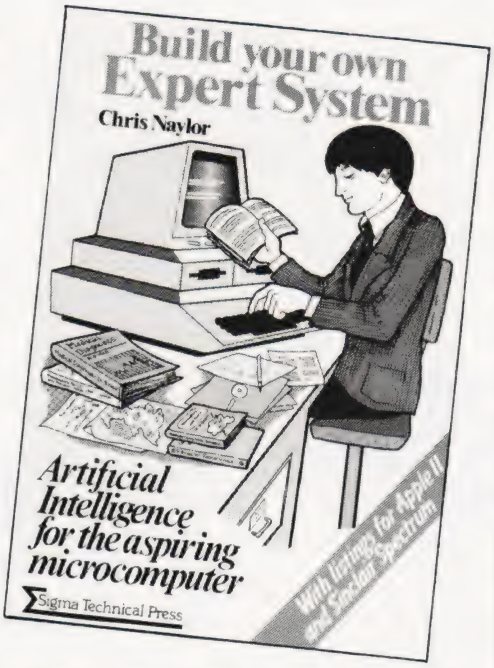




The FORTH book is also a good, even essential source of reference for FORTH on the BBC Micro. Unfortunately, despite a length of almost 300 pages, it contains few examples programs for applications of any consequence, so that it is short on motivational material which is particularly necessary with a language as unreadable as FORTH. This FORTH book was mentioned by D. S. Peckett in his article 'Going FORTH Again' in the July 1983 issue of **Computing Today** as suffering from production delays. Acornsoft assure us that their production problems have now been solved and that the book, and the language itself, are now readily available. Despite this, I cannot recommend that you rush out and buy it, even if you are a FORTH enthusiast with a BBC Micro, as it is much too stodgy for my taste.

To show that it is possible to write an interesting book on programming in a language other than BASIC for a specific micro, it is only necessary to turn to **Pascal Programs for Games and Graphics in UCSD Pascal for the Apple** by Tom Swan. Here we have a series of programs for imaginative games written in Pascal with a beautifully clean and clear style. Thus, at the same time we have a mine of ideas for games and for graphics (since all the programs make good use of graphics) as well as a practical primer on the writing of well-structured programs. Even if you don't have a Pascal system, the book is worth considering, for the programs are so well written that the task of translating them to BASIC is as easy as it can be. A suite of programs for a graphics editor is also given: that is, for a system to handle graphics in much the same way as a word processor handles text. Converting this for your system could also be a rewarding task. In this way, the book is a good one for its target audience, but it is also of value to others. Can we have more books like this please?

Also, can we have more along the lines of **Build Your Own Expert System** by Chris Naylor? I have to admit that I don't much care for his style of presentation or his sense of



humour, and that I would rather not have had to plough through his first two chapters. The first, 'Why expert systems?' takes 12 pages to give the answer 'Because that is what the book is about'. The second, 'A statistical scheme' is a treatment of basic probability theory, not an attractive topic, but even less so when we see Chapter 3 is called 'Avoiding probabilities'! However, I can forgive more than this for the idea behind the book which is to show us how to write our own expert system on our own micro. It includes program listings for the Spectrum and Apple II, and although there is an error in the third line of the very first program, this is fortunately not typical of all the programs in the book. In fact, all the Spectrum programs are reproduced from listings on the Sinclair printer and so can be presumed to be accurate.

The last two books are the kind I, for one, should like to see more of. By demonstrating novel and imaginative ways of programming and using micros, they can only help the progress of the whole micro scene.

This month's books are:

The Fifth Generation by Edward A. Feigenbaum and Pamela McCorduck, Michael Jospeh, 275 pages, £9.95.
Beyond BASIC by Richard Freeman, BBC and NEC, 256 pages, £7.25.
LISP on the BBC Microcomputer by Arthur Norman and Gillian Cattell, Acornsoft, 197 pages, £7.50.
FORTH on the BBC Microcomputer by Richard de Grandis-Harrison, Acornsoft, 280 pages, £7.50.
Pascal Programs for Games and Graphics by Tom Swan, Hayden, 214 pages, £15.95.
Build Your Own Expert System by Chris Naylor, Sigma Technical Press, 249 pages, £6.95.

LORDS OF TIME

Joins our range of acclaimed pure-text puzzle adventures, at £9.90, for:

BBC 32K ***COMMODORE 64*** ***SPECTRUM*** 48K ***LYNX*** 48K ***NASCOM*** 32K ***ORIC*** 48K ***ATARI*** 32K

ADVENTURE REVIEWS

"Adventures which have a fast response time, are spectacular in the amount of detail and number of locations, and are available to cassette owners . . I am extremely impressed . . The Level 9 Adventures are superbly designed and programmed, the contents first rate. The implementation of Colossal Cave (Adventure) is nothing short of brilliant; rush out and buy it. While you're at it, buy their others too. Simply smashing!"
– SOFT, Sept 83

"I found Dungeon exceedingly well planned and written, with a fast response. There are well over 200 locations and the descriptions are both lengthy and interesting. The objects number about 100. It could therefore take some months to explore the whole network, giving many hours of enjoyment in the process."
– C&VG, Sept 83

"The descriptions are so good that few players could fail to be ensnared by the realism of the mythical worlds where they are the hero or heroine . . .great fun to play."
–Which Micro?, Aug 83

"My appetite has been whetted and I intend to get my own copy (of Snowball) to play."
– What Micro?, Dec 83

ADVENTURE REVIEWS

"This has to be the bargain of the year. If adventures are your game then this (Colossal Adventure) is your adventure."
– HCW, 5 Sept 83

"Colossal Adventure is simply superb. Anyone who wishes to use adventures in an educational setting really must use and see this program as it emulates Crowther and Wood's masterpiece so well. For those who wish to move onto another adventure of similar high quality, Dungeon Adventure is to be recommended. With more than 200 locations, 700 messages and 100 objects it will tease and delight!"
– Educational Computing, Nov 83

Colossal Adventure is included in Practical Computing's Top 10 games choice: "Poetic, moving and tough as hell."
– PC, Dec 83

"To sum up, Adventure Quest is a wonderful program, fast, exciting and challenging. If you like adventures then this one is for you"
– NILUG # 1.3

"Colossal Adventure . . For once here's a program that lives up to its name . . a masterful feat. Thoroughly recommended"
– Computer Choice, Dec 83

"wholly admirable"
– Your Computer, Sept 83

MIDDLE EARTH ADVENTURES

1: COLOSSAL ADVENTURE
A complete, full size version of the classic mainframe game "Adventure" with 70 bonus locations added.

2: ADVENTURE QUEST
Centuries have passed since the time of Colossal Adventure and evil armies have invaded The Land. The way is long and dangerous; but with cunning you can overcome all obstacles on the way to the Black Tower, source of their demonic power, and destroy it.

3: DUNGEON ADVENTURE
The trilogy is completed by this superb adventure, set in the Dungeons beneath the shattered Black Tower. A sense of humour is essential!

THE FIRST SILICON DREAM ADVENTURE

1: SNOWBALL
The first of Pete Austin's second trilogy. The giant colony starship, Snowball 9, has been sabotaged and is heading for the sun in this massive game with 7000 locations.

THE LORDS OF TIME SAGA

7: LORDS OF TIME
Our congratulations to Sue Gazzard for her super design for this new time travel adventure through the ages of world history. Chill to the Ice-age, go romin' with Caeser's legions, shed light on the Dark Ages etc. etc. We'll be selling this game mail-order from January 1st

---

Price: £9.90 each (inclusive)

Level 9 adventures are available from good computer shops, or mail-order from us at no extra charge. Please send order, or SAE for catalogue, to

LEVEL 9 COMPUTING

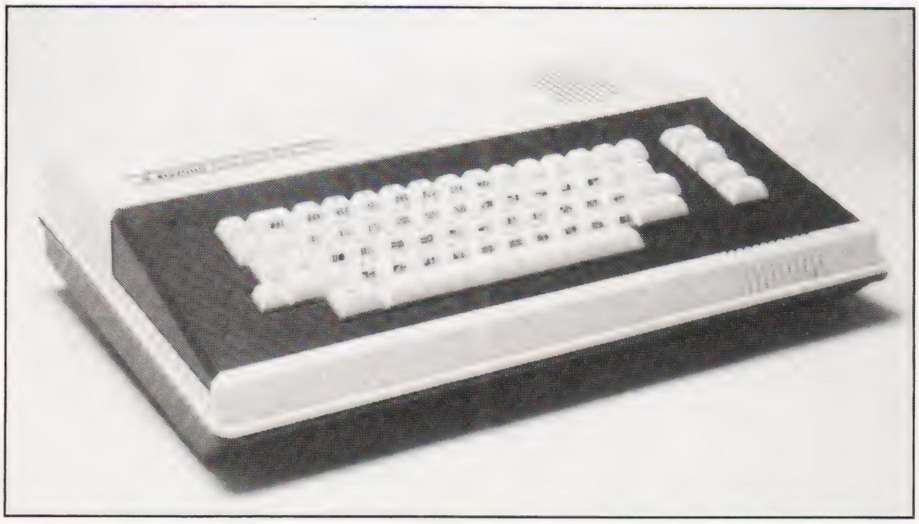
Dept C, 229 Hughenden Road, High Wycombe, Bucks HP13 5PG

Please describe your Computer

John Fairbairn

COLOUR GENIE MONITOR

G-Mon is a comprehensive machine code monitor written for the Colour Genie. Now there's no excuse for not getting to grips with your Z80.



Like many people who learnt Z80 machine language on the Video Genie/TRS 80 system, my first monitor was T-Bug. In my innocence I used this for months, thinking it must be the bee's knees as it was produced by Tandy. As my experience grew I became frustrated with it, and was pleased to see patches in computer magazines for some of the 'bug'-bears I had encountered. Eventually it dawned on me I should get another monitor.

I have been through several now, but none of them seems to give me what I — and I believe most micro users — really need.

Most monitors seem to be digests of systems written for mainframes and minis and we all know that people who work on these machines ***never*** make mistakes. But people like me do type J 5323 instead of J 5332, and our programs do jump to non-existent points and disappear into the wide blue yonder. People like me are always forgetting to fix breakpoints after setting them, and with the constant gnashing of our teeth as we relentlessly type our programs back in after the inevitable crash, we must all be looking more each day like the Laplanders who chew leather for a living.

But I lived lazily with all these problems until recently my Video Genie ground to a halt. This was a marvellous opportunity to upgrade and I didn't have to stop and think what to buy. All the books and tapes I'd bought for the VG couldn't be allowed to go to waste, so it had to be the only compatible machine, the Colour Genie.

Cutting a sob story short, it was nothing like as compatible as I'd been led to believe, at least where machine code is concerned. Conversion is possible, but only if you can get at the machine code — and I couldn't get at that as I couldn't get my monitor in. Before apathy set in and commercial monitors appeared in the shops, I decided to write my own monitor and to get rid of all those niggles I'd had in the past. The result is the program here. It is in BASIC, of course, as that is the only way to get it into the Genie without another monitor, but once it is in you can save it as a SYSTEM tape. It occupies only 770 bytes at the top of memory.

LOADING G-MON

On power-up you should protect high memory for the machine code by answering MEM SIZE? with 30000. Then type the program in and CSAVE it, just in case. Then RUN the BASIC program (it takes about 12 seconds). Nothing visible happens — all that ***is*** happening is that the values READ in by the DATA statements are being put, in hexadecimal form, into high memory in the area from &H7CD0 to &H7FD1.

When the prompt returns you no longer need the BASIC program, except as a back-up. If you have entered the data correctly you should get a checksum of 86325. You should get a cleared screen and a message saying G-Mon 1.1 followed by a prompt, $ in this case. Now you should check all the commands are operating perfectly, so let's run through them.

G-MON COMMANDS

Type **S** and you will return at once to BASIC, with your high memory still protected. To return to G-Mon from BASIC simply CALL 7CD0.

Type **D** (the comuter responds with D:) and then put in a four-figure hexadecimal number (eg 0000 will give the start of the ROM, 7CD0 the start of G-Mon) and as soon as you have typed the fourth digit the comuter will display (or dump) a screenful of memory, neatly formatted for easy checking (128 bytes at a time). Then the prompt $ asks for a new command. This is one of the facilities lacking in T-Bug, by the way (see Photo 1).

Type **R** next and the computer will display the values of the registers (see Photo 2). You obviously have to be familiar with Z80 machine language to make sense of this. Capitals are used for the names of the main register set, lower case for the alternate registers. SP stands for Stack Pointer and BP for Break Point (ie the location of the last entered break point). The values shown are those after the last machine code instruction, which can include some operations caried out by G-Mon on returning from your program. This command is normally meant to be used in conjunction with a break point, in which case it is called automatically and the register values shown are those at the time of the break point with no interference from G-Mon.

Now try **M**. The computer responds with M: and you then have to give it the four-figure hex address at which you want to start putting data into memory. This is the machine-code way of doing what our BASIC program did. G-Mon then repeats the address and shows its current contents. You can then insert data at that point. Each entry must be two hex digits. As soon as you have typed the two digits that value is stored in memory

at that point and the next address is printed. To stop inserting data type **X** at any time and you will return to the prompt $. To move forward or back through memory, without affecting the memory contents, type either ↓ for forward or ↑ for back. In the latter case ★ will be printed to warn you you are going back (this back-stepping is another feature not found in T-Bug).

Note that the Repeat key will work after all these commands.

You cannot insert data into ROM (if you try it is just ignored) and of course you should avoid putting anything in the area used by G-Mon: 7CD0 to 7FFF (7FD2 to 7FFF is its stack area).

To see M at work, type M:47BF and after the 47BF:20 prompt type 51. You should see Q at the bottom right of the screen (the screen memory occupies 4400 to 47BF). You can also see what you've inserted by running through memory with M or using D for dump.

Try a jump command now: type **J**. This tells the computer to start execution of a program. For the start address you are prompted by J: and you have to insert a four-figure hex address. If you make a mistake you can't backspace. In the case of M and D this doesn't matter, but in the case of J (and T-Bug) the results could be catastrophic if the computer acts on your mistake. Therefore, as a safety measure, once you have put the address in you have to type J again to confirm execution can begin. Typing any other key aborts the instruction and returns you to the prompt $. Then you can try again.

Try J:1A19J and you will return to BASIC (equals the S command). Or you can try J:7CD0J and you will jump to the start of G-Mon, as your screen display will testify. Other jumps have to be controlled, so you must write your program first.

Remember, incidentally, all programs you write should return to a control point when you are debugging them. In this case they could end with a jump back to G-Mon (C3 D0 7C).

BREAKING OUT

Often when you run a program you want to be able to stop it somewhere, typically to check the state of the registers and memory at that point to ensure the programs has no bugs so far. For this you insert, with **B**, a breakpoint (like STOP in BASIC) at the address where you want to stop. This has to be a sensible one — you can't stop in the middle of an instruction — but if you've done it properly, when you run your program (with a J) the program stops as soon as that address is encountered by the program counter.

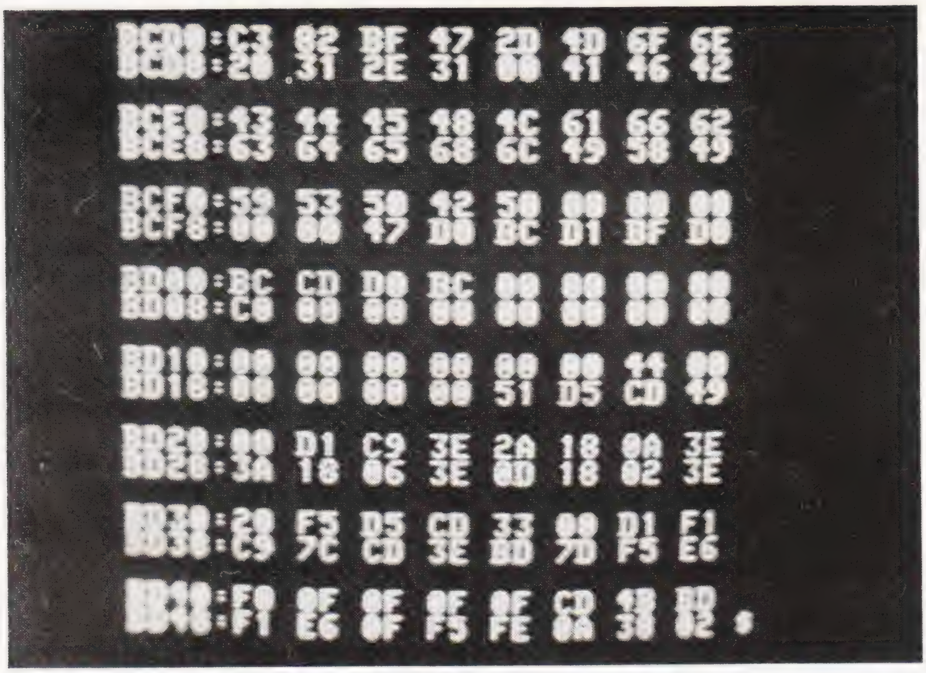
G-Mon does three things in that case: it prints out the state of the registers at the time of the breakpoint, it returns the user to the prompt $, and it repairs the breakpoint. This means you do not have to press F to fix the break point, as in other monitors. I have also done away with the G (for Go) command as I have never used it. For single-step breakpointing through a program I use another, much larger utility.

Break is one of those instructions that can play havoc with a program if not done right. Therefore when you type B and are prompted with B:, you insert your hex address in the usual way but have to type B again to confirm it (any other key aborts the instruction). You will then be prompted with J: to ensure you do perform the program and clear the breakpoint. You then type in the starting address and J and on return to $ (if your program flow is not faulty and has not avoided the break address) the break point will have cleared.

Note that you can't set a break point in the ROM.

To try this command, first put the following instructions into memory with the M command starting at address 6000: 21 CC 45 3E 51 77 C3 D0 7C. Then type B:6006B and in response to J: type 6000J. You should see a little flash in the middle of the screen and the registers will be printed. The only ones you used were AF and HL and these should read 5100 and 45CC respectively.

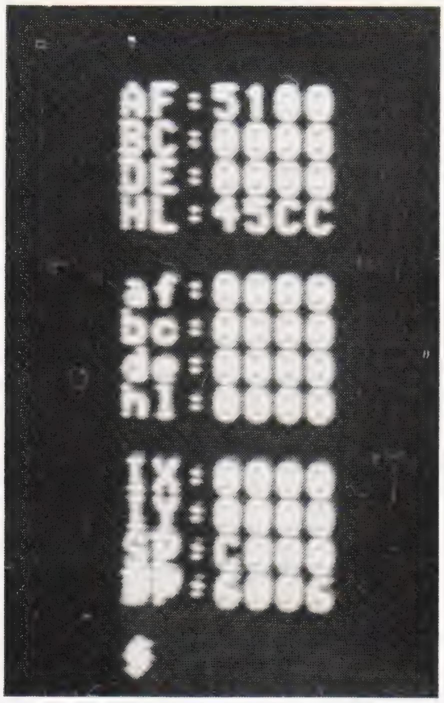
But what was that little flash? One of the things that most intensely annoyed me about my previous monitors was that on return to the prompt they wiped out what was on the screen. If I was debugging a graphics or text display, that wasn't a lot of help. So G-Mon has a Pause command. By hitting **P** you insert a pause just before the machine returns from the break point. To see this in action, type P. An ★ will be printed as a warning then type in the same break point and jump point as above (6006 and 6000). The ★ is repeated after the B instruction. On this occasion you should see that in the middle of the screen is a Q. The display will pause there until you hit any key, then you return to $ with the breakpoint *and* the pause cleared.



TAPE STORAGE

If the point you jump from does not pass over the break point you have set you have lost control and can suffer irreversible damage. If you are worried that might happen and don't want to retype your program, the next command is the crucial one. **W** for write writes a SYSTEM tape. After W you will get W:. You now have to insert three addresses in the right order: the starting address of the area you want to copy; then the end address of the area you want to copy (inclusive); then the address at which execution is to begin (this may be in another area).

Then you have to give your saved file a name. Since the SYSTEM command reads only the first letter of a name it is pointless giving it a long name, so only one



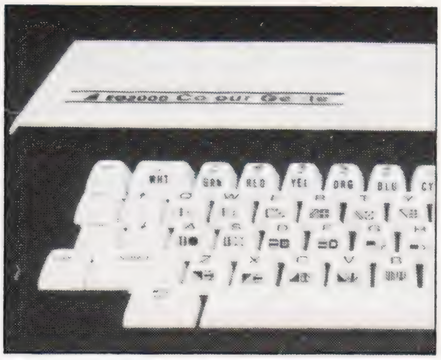
letter is allowed. Type that letter, which is then displayed. Then a final check — if any of your input is wrong hit any key except W and the instruction will be aborted. If you are happy, switch on your cassette to record and hit W. An ★ will be displayed to warn you recording is taking place. When recording is finished the prompt $ returns (you have to switch off the cassette now).

The first priority is to save G-Mon as a SYSTEM tape. The full instruction for this is W:7CD0 7FD1 7CD0 GW. You will then have created a SYSTEM tape with file name G. To load this, or any other SYSTEM tape, go back to BASIC, type SYSTEM and after the prompt ? type the file name. Switch on the cassette. When the file is found and being loaded, asterisks blink on and off in the usual way. On completion of loading you will be prompted with ? again and can load more programs (it is your responsiblity to see they don't overlap). G-Mon loads in about four seconds.

When all loading is finished answer ? with / then the decimal address of the point you want start execution, or type / alone and execution will start at the address specified for the last loaded tape. At any future point if you RESET, you can get back to your machine-code programs in the same way without re-loading.

If you are already in G-Mon and want to load another tape, you can call the SYSTEM command up from the ROM by means of the instruction J:02B2J.

If you are working with BASIC and G-Mon together, it is your responsibility to set sufficient area free at power-up to allow room for all your machine code programs above BASIC. Those not working in BASIC can use memory from 5800 up. The low-res graphics screen starts at 4400 and the high-resolution display area starts at 4800.



```
01 02 03 21 D1 7F 11 D1
BF ED B8 06 04 3E 7C 1E
BC 21 D0 BC C5 01 02 03
BE 20 01 73 23 0B 0C 0D
20 F6 04 05 20 F2 C1 3C
1C 10 E6 C3 D0 7C
```

Listing 1. Colour Genie patch.

CONVERSIONS

G-Mon is not suitable for other machines because it uses system-dependent code, naturally. But 32K Colour Genie owners can relocate it to the top of 32K memory by loading as above then running the program in listing 1 which should be loaded at 6000, using M. Run it with J:6000 J then, with M, insert the following data at the addresses specified:

BDD9: C0	BE83: 7E
BDF3: C0	BE88: 7E
BD91: 7E	BEAB: 7E
BD9D: 7E	BF2B: 7E
BDAB: 7E	BD3D: 7D
BDFE: 7E	BF4F: 7D
BE1B: 7E	BD39: 7C

You can now save the revised G-Mon on tape with W:BCD0 BFD1 BCD0 GW.

Listing 2. BASIC listing for G-MON.

```
10 FOR I=&H7CD0 TO &H7FD1
11 READ A: POKE I,A
12 T=T+A
13 NEXT
14 PRINT "Data checksum =";T
15 END
16 '
17 DATA 195,130,127,71,45,77,111,110,32,49,46,49,0,65,70,
   66,67,68,69,72,76,97,102,98,99,100,101,104,108,73,88,73
18 '
19 DATA 89,83,80,66,80,0,0,0,0,0,71,208,124,209,127,0,112,
   205,208,124,0,0,0,0,192,0,0,0,0,0,0,0
20 '
21 DATA 0,0,0,0,0,0,68,0,0,0,0,0,81,213,205,73,0,209,201,
   62,42,24,10,62,58,24,6,62,13,24,2,62
22 '
23 DATA 32,245,213,205,51,0,209,241,201,124,205,62,125,125,
   245,230,240,15,15,15,15,205.75,125,241,230,15,245,254,10,
   56,2
24 '
25 DATA 198,7,198,48,205,49,125,241,201,254,48,216,254,58,56,
   9,254,65,216,254,71,48,5,214,7,214,48,201,55,201,205,29
26 '
27 DATA 125,205,89,125,56,248,24,211,205,89,125,216,24,205,
   205,129,125,205,110,125,7,7,7,7,79,205,110,125,129,119,
   43,201
28 '
29 DATA 205,126,125,205,47,125,16,248,201,205,39,125,205,126,
   125,42,255,124,205,43,125,205,57,125,205,39,125,126,205,
   62,125,205
30 '
31 DATA 47,125,205,29,125,254,10,40,34,254,91,40,33,254,88,
   202,146,127,205,120,125,56,235,7,7,7,7,71,205,29,125,254
32 '
33 DATA 88,202,146,127,205,120,125,56,243,128,119,35,24,196,
   205,35,125,43,24,190,205,39,125,6,1,205,144,125,205,201,
   1,42
34 '
35 DATA 255,124,30,128,14,2,205,57,125,205,39,125,6,8,126,
   205,62,125,205,47,125,35,29,202,149,127,16,242,205,43,
   125,13
36 '
37 DATA 32,228,205,43,125,24,221,205,39,125,205,126,125,205,
   29,125,254,74,194,146,127,237,115,7,125,49,9,125,217,8,
   253,225
38 '
39 DATA 221,225,225,209,193,241,217,8,225,209,193,241,237,
   123,7,125,42,255,124,233,245,213,205,73,0,209,241,237,
   115,7,125,49
40 '
41 DATA 29,125,245,197,213,229,8,217,245,197,213,229,221,
   229,253,229,217,8,237,123,7,125,205,201,1,33,28,125,17,
   221,124,14
42 '
43 DATA 3,6,4,205,43,125,26,205,49,125,19,26,205,49,125,19,
   205,39,125,126,205,62,125,43,126,205,62,125,43,16,228,205
44 '
45 DATA 43,125,13,32,220,237,91,5,125,33,1,125,1,3,0,237,176,
   195,146,127,205,39,125,33,6,125,205,126,125,205,29,125
46 '
47 DATA 254,66,194,146,127,42,5,125,17,1,125,1,3,0,237,176,
   43,58,4,125,167,40,12,151,50,4,125,17,68,126,205,35
48 '
49 DATA 125,24,3,17,75,126,114,43,115,43,54,195,205,43,125,
   62,74,205,49,125,33,0,125,195,23,126,50,4,125,205,35,125
50 '

51 DATA 195,146,127,205,39,125,6,3,205,144,125,205,29,125,
   205,49,125,119,205,29,125,254,87,194,146,127,205,35,125,
   205,63,2
52 '
53 DATA 221,33,250,124,42,253,124,237,91,255,124,237,82,35,
   62,102,205,31,2,62,85,205,31,2,6,6,221,126,0,205,31,2
54 '
55 DATA 221,43,16,246,37,250,70,127,62,60,205,31,2,151,205,
   31,2,205,106,127,24,238,151,189,40,12,62,60,205,31,2,125
56 '
57 DATA 205,31,2,205,106,127,62,120,205,31,2,237,75,251,124,
   121,205,31,2,120,205,31,2,195,146,127,71,123,205,31,2,122
58 '
59 DATA 205,31,2,131,79,26,205,31,2,129,79,19,16,247,121,
   195,31,2,205,201,1,49,0,192,33,211,124,205,167,40,151,50
60 '
61 DATA 4,125,205,43,125,33,0,125,62,36,205,49,125,205,29,
   125,205,49,125,254,68,202,228,125,254,77,202,153,125,254,
   66,202
62 '
63 DATA 164,126,254,74,202,23,126,254,80,202,234,126,254,82,
   202,75,126,254,87,202,243,126,254,83,202,25,26,62,8,205,
   49,125,24,195
```

Paul Gardner and Caroline Bradley

LEARNING FORTH PART 4

Last month I dealt with the structures that FORTH uses to control the flow of execution within a program and also began to explain how various dictionary entries are stored and how you can create your own 'defining words'.

We'll now see how we can use <BUILDS . . . DOES> to produce a word which can define an array. For the first example we want a word ARRAY which, when used as follows:

```
n ARRAY name
```

will set up a dictionary entry called 'name' which will be an entry of n+1 elements (0 to n), so that when 'name' is used, for example:

```
m name
```

it will return the memory address of the mth element of the array. (I'm using arrays with subscripts 0 to n as this will be familiar to most BASIC users.)

If this all sounds a little confusing then the idea compares to the use of BASIC's DIM function, comprising DIM name (n) which sets up the array, and name (n) which lets you access any element of the array. So we need a word whose actions are:

(define-time action) Allocate space in the dictionary for the array.

(run-time action) Given a number, m, on the stack, return the physical address of the mth element of the array.

Listing 1 gives a definition for the word ARRAY which will do what I have described. It could be used, for example, as:

TAKE IN CRA

This would set up an array called COUNTERS with 21 elements (0-20). Using:

```
10 COUNTERS
```

would give the address of element 10, so that:

```
10 COUNTERS @
```

would return the number held in this element of the array, and:

```
14 10 COUNTERS !
```

would store the number 14 in element 10 of the array.

The way the define-time action of the word works is as follows. <BUILDS sets up the dictionary header for the new word, the expression 1 + 2 ★ ALLOT adds one to the number on top of the stack, multiplies the result by two and sets aside (ALLOTs) that many 'bytes' of memory in the dictionary for the parameter field of the new word. Remember, given a number n, we want n+1 elements which take up 2★(n+1) bytes of memory as each number takes two bytes.

ALLOT works in much the same way that C, and , can be used to enclose one and two bytes of memory in the dictionary but ALLOT can set aside as many as necessary and also does not store any numbers in these bytes, it just makes the space available.

The way the run-time action works is quite simple. Assuming you have used your new array to store some numbers, then whenever you want to access the array (ie get a number from it or put a number in it) you use an expression like

```
7 COUNTERS
```

The word COUNTERS leaves the address of its parameter field (the actual array space) on the stack and calls the run-time action of its defining word ARRAY to use this number. The expression in ARRAY:

```
SWAP  2  *  +
```

calculates the address of that particular element of the array.

As this is the first example I'll run through this run-time action bit by bit. When the run-time action of ARRAY is called there are two numbers on the stack (these are the required element (10) and the address of the zeroth element of the array (COUNTERS)).

The expression SWAP 2 ★ leaves on the stack (address of the zeroth element) and (offset in bytes to required element). Then + leaves on the stack the actual address of the required element, which can be used by @ and ! to fetch a value stored or to store a new one.

The word ARRAY can be used to define any number of unique arrays with any number of elements (memory permitting) in the same way that VARIABLE can create lots of different variables.

CHECKING IT OUT

While the definition of ARRAY may seem particularly short for such a powerful command, it does have serious drawbacks. When a new array is defined it does not initialise the contents: more seriously, when the array is used there is no error-checking to see if what you are trying to do is 'legal'.

For example in Spectrum BASIC, if you set up an array using:

```
DIM p(10)
```

and then try:

```
LET a = p(12)
```

the program will stop with a "Bad subscript" error message.

It is very important to make sure that you stay within the limits of your array, because if you change the contents of a memory address just outside it, you will corrupt the dictionary entry of an adjacent word. This usually means that sooner or later your program will 'crash' seriously. Listings 2 and 3 give definitions for a few words that overcome these difficulties.

The new defining word ARRAYCHECK in Listing 3 is a word which will set up arrays in the dictionary but will initialise all the elements of the array to zero at define-time, and at run-time will check that you are attempting to access a valid element.

To explain, at define time the define-time action takes a number of the stack (call it n) and adds one to it; this is the number of elements. This number is duplicated and stored in the first two bytes of the parameter field. The same number is then used to control the upper limit of a DO . . . LOOP which repeatedly encloses the number 0 in the dictionary by

```
( LISTING 1 - DEFINING WORD FOR ONE DIMENSIONAL ARRAYS )
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A )

: ARRAY <BUILDS ( N - )
  1+ 2 * ALLOT
   DOES> ( N,ADDR-ADDR )
   SWAP 2 * + ;
```

Listing 1

using , (comma). For example, if we set up an array using:

```
3 ARRAYCHECK TEST
```

then the define-time action of ARRAYCHECK would set up a dictionary entry like the one shown in Fig. 1. The first two bytes in the parameter field contain the number of elements in the whole array and the rest of the parameter field is initialised to contain zeros.

Error-checking, of course, makes the run-time action of ARRAYCHECK a little more complicated, so I have defined two words CHECKBOUND and ERRORMESSAGE to make the definition of ARRAYCHECK clearer. Whenever our new array TEST is used, it calls the run-time action of ARRAYCHECK. At this time the top two numbers on the stack will be the element of TEST we require and the address of TEST's parameter field.

CHECKBOUND uses these two numbers to fetch the contents of the first two bytes of this parameter field. This contains the number of elements in the array or, to put it another way, the upper limit plus one. CHECKBOUND returns a value to the top of the stack which is one, ie true, if the required element is (greater than or equal to zero) AND (less than or equal to the upper limit).

If CHECKBOUND returns 'true' then ARRAYCHECK carries on and works out the address of the required element of the array as before. (The 2 + is to take into account the two bytes taken up to hold the count for the number of elements.)

If CHECKBOUND returns 'false' then ERRORMESSAGE is called, which uses the top two numbers on the stack (ie required element, parameter field address) to produce a suitably useful message. Execution of any program running is then stopped. Most programs in FORTH would probably use ABORT to abort the program and clear the stack, but in Abersoft's version ABORT also clears the screen so that your error message would promptly disappear. So I have used the expression SP! QUIT to clear the data stack and quit execution of the current program.

To see how this works in practice, try:

```
3 ARRAYCHECK TEST
```

and then attempt to use an element that is out of bounds, for example:

CRA 15

The computer will respond:

```
ERROR! Value 6 out of bounds
Range allowed is 0-3
ok
```

Clever, eh?

2-D ARRAYS

Listing 4 gives the definition of a word that will create two dimensional arrays (ie rows and columns). This definition is a non-checking, non-initialising one. At first sight this definition of 2-D seems more complex than that of ARRAY, but it is essentially the same. The word 2-D is used in the form:

```
row-limit  column-limit  2-D  name
```

For example:

```
8  8  2-D  CHESSBOARD
```

Any element of the array would be accessed using:

```
row  column  name
```

For example:

```
4  2  CHESSBOARD
```

The define-time action of 2-D first of all has to save in the parameter field of the new word the number of columns in the array. Then the total number of bytes which the array will occupy is calculated:

```
number of bytes = 2*((limit for
rows + 1)*(limit for columns + 1))
```

and these are set aside in the dictionary using ALLOT as defined earlier.

The run-time action of 2-D expects three numbers on the stack (row, column, parameter field address of new array), and it calculates the address of the required element. All the elements of the array are stored in order in the dictionary; for example, if we had defined an array using:

```
2  3  2-D  EXAMPLE
```

the elements would be stored in the dictionary in the following order:

```
(0,0),(0,1),(0,2),(0,3),(1,0),
(1,1), . . . . . . . . . ,(2,3)
```

so that to calculate where the required element is in the list you work out:

```
((required row)*(number of
columns) + required column
```

For example, in the above 2-D array element (1,1) is:

```
(1*4) + 1 = 5th element
```

Remember our list begins with the zeroth element, so the offset in bytes from the parameter field address is in this case (2★5) plus the two bytes used to store the number of columns.

I have included in all my definitions comments at the right hand side to indicate the contents of the stack at various points.

Listing 5 gives a defining word 2-DCHECK which operates similarly to 2-D, but initialises the elements of the array to zero when the array is defined and at run time checks that both subscripts (row and column) are within the limits of the array.

In this case the define-time action of 2-DCHECK needs to store in the parameter field of the new word both the number of rows and the number of columns in the array. The run-time action of 2-DCHECK is essentially similar to that of ARRAYCHECK, but in this case we need to check for both row and column limits. Figure 2 gives a diagrammatic view of the dictionary entry for a word defined as:

```
3  4  2-DCHECK  BOARD2
```

This array has (3 + 1) ★ (4 + 1) = 20 elements, and the number of columns and the number of rows are stored as the first two numbers (four bytes) in the parameter field of BOARD2.

So, the run-time action of 2-DCHECK uses the numbers provided for the required row and column, along with the parameter field address of the array, as parameters to pass to CHECKBOUND to see if the subscripts are in range. If either subscript is wrong then a suitable error message is printed as before, only this time I have arranged that 2-DCHECK lets you know whether the mistake is in the row or column.

HERE BE MONSTERS

Listings 6 and 7 show how flexible the use of defining words is in constructing your own dictionary entries. This example sets up a somewhat larger and peculiar array for holding

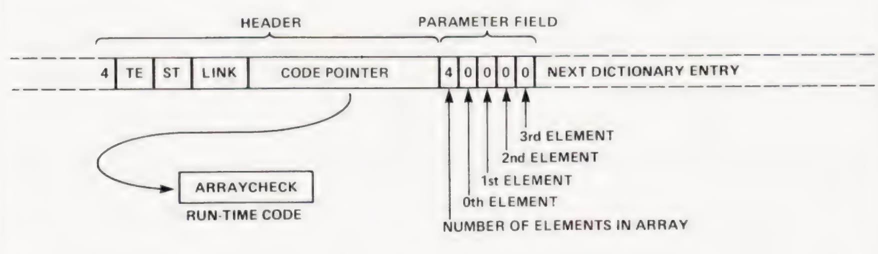


Fig. 1 A typical ARRAYCHECK dictionary entry.

```
( LISTING 2 - ASSOCIATED WORDS FOR SELF CHECKING ARRAYS)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)

: CHECKBOUND ( N,ADDR-N,ADDR,FLAG)
  OVER OVER @  ( N,ADDR,N,UL+1)
  OVER >       ( N,ADDR,N,[UL+1>N])
  SWAP -1 >    ( N,ADDR,[UL+1>N],[UL>=0])
  AND          ( N,ADDR,FLAG[FLAG=1 IF VALID ELEMENT])
;

: ERRORMESSAGE ( N,ADDR-)
  CR SWAP ." ERROR! Value " . ." out of bounds."
  CR ." Range allowed is 0-" @ 1 - . CR
;
```

Listing 2

```
( LISTING 3 - DEFINING WORD FOR SELF CHECKING 1-D ARRAYS)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)

: ARRAYCHECK <BUILDS ( N-)
    1+ DUP , 0 DO 0 , LOOP
   DOES> ( N,ADDR-ADDR )
    CHECKBOUND IF SWAP 2 * + 2+ ( ADDR)
    ELSE ERRORMESSAGE SP! QUIT THEN ;
```

Listing 3

```
( LISTING 4 - DEFINING WORD FOR 2-D NON-CHECKING ARRAYS)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)

: 2-D <BUILDS ( NUM. OF ROWS,NUM. OF COLS -)
      1+ DUP , SWAP 1+ * 2 * ALLOT
     DOES> ( ROW,COL,ADDR-ADDR)
      ROT OVER @ ( COL,ADDR,ROW,NO.OF COLS)
      * ROT +    ( ADDR,ELEMENT)
      2 * + 2+   ( ADDR)
;
```

Listing 4

```
( LISTING 5 - DEFINING WORD FOR 2-D SELF CHECKING ARRAYS)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)

: 2-DCHECK <BUILDS ( NUM.OF ROWS,NUM.OF COLUMNS)
    1+ DUP , SWAP 1+ DUP ,    ( COLS+1,ROWS+1)
    * 0 DO 0 , LOOP
   DOES> ( ROW,COL,ADDR-ADDR)
    CHECKBOUND                ( CHECKS IF COL IN RANGE )
    IF ROT SWAP 2+            ( COL,ROW,ADDR+2)
     CHECKBOUND               ( CHECKS IF ROW IN RANGE)
     IF 2 - DUP @             ( COL,ROW,ADDR,NO.OF COLS)
      ROT * ROT + 2 * + 4 +   ( ADDR OF REQUIRED ELEMENT )
     ELSE ERRORMESSAGE ." for row of array." SP! QUIT
    THEN
   ELSE ERRORMESSAGE ." for column of array." SP! QUIT
  THEN ;
```

Listing 5

the names and attributes of 19 different monsters that wish to do battle with you in my FORTH version of that famous game THE-VALLEY.

While the details of the defining word MARRAY will not be clear until the end of this series of articles (as it relies heavily on moving strings of text about before it encloses them in the dictionary) the general idea is quite instructive. The defining word MARRAY has a define-time action which requires no values upon the stack, but instead prompts the user to type in the details for the four different fields of this array. When used as:

```
MARRAY MONSTERS
```

you can type in repeatedly the names, physical strenths and psi strengths of your monsters along with a code letter which determines the scenes that a particular nasty creature can be present in. All this information is thus stored in the array MONSTERS which is set up to contain (diagrammatically):

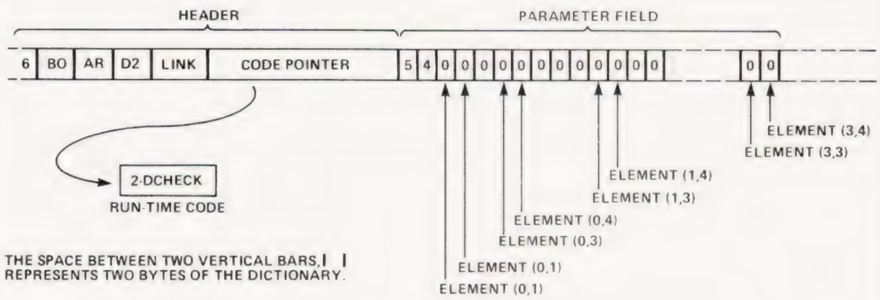


Fig. 2 A typical 2-DCHECK dictionary entry.

The run-time action of the word MARRAY takes two numbers off the stack (required element (0-18), required field (0-3)), for example:

```
4  MNAME  MONSTERS
```

where we have defined MNAME, MSTRENGTH etc as constants to make the program clearer. The run-time action returns either the required address for MSTRENGTH, MPSI and MCODE or an address and a number for MNAME (address of first letter of monster's name and a count of the number of letters in its name). This allows the name to be printed using the FORTH word TYPE.

name0 (15 letters max)	strength0 (0-255)	psi-strength0 (0-255)	code-letter0 (1 letter)
name1	strength1	psi-strength1	code-letter1
name2	strength2	psi-strength2	code-letter2
.			
.			
.			
name18	strength18	psi-strength18	code-letter18

Next month I shall be dealing with input and output, which should clarify the details of this defining word and allow us to produce something like it for names and addresses, dates of birth and so on.

A RANDOM WALK

It is quite difficult in such a series as this to provide numerous small examples to show the use of a particular feature of a language, so the next few listings demonstrate, using fairly complex but nicely compatible

programs, many of the aspects of FORTH that I have covered so far.

Listing 8 is a few words to provide a random number generator, which is missing from FORTH. The generator uses the same method as the built-in RND function on the Spectrum, and the mathematical method is described fully on pages 56 and 57 of the manual. To use the random function type:

```
n  RND
```

where n is a positive integer, and a random number between zero and n−1 will be returned to the stack. Like the BASIC version you can repeat a given series of random numbers by using:

```
n  RAND
```

This initialises the seed for the generator.

One particular feature of RAND is that if you type:

```
Ø  RAND  32
```

the program uses the Spectrum clock to initialise the seed.

```
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
H*******************************H
HHH.HHH.H.HHH.HHHHHHH.HHHHHHH*H
H...H.H.H...H.H.....H.H......*H
H.HHH.H.HHHHH.H.HHH.HHH.H.HHH*H
H.H...H.H...........H...H.H.H*H
H.H.HHH.H.HHHHH.HHH.H.HHH.H.H*H
H.H.......H...H...H.H...H*****H
H.HHHHH.HHHHH.HHHHH.HHHHH*HHHHH
H.....H.H.H......*********HoooH
H.HHHHH.H.HHH.HHH*HHHHHoHHHoHHH
H.H.........H...H*HoHoooHoooHoH
H.HHH.HHH.HHH.H.H*HoHHHHHoHHHoH
H...H.H...H...H***ooooooooHoooH
H.H.HHHHH.H.HHH*HHHHHoHHHoHoHHH
H.H.......H.H.H*HoooooH.HoooH.H
H.HHH.H.HHH.H.H*HHHoHHH.HHHoH.H
H...H.H.H...H..***HoH.....HoH.H
HHHHH.HHH.H.HHHHH*HHH.HHH.HHH..
H.........H......***************
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
```

Fig. 3 Typical maze and route out.

Listings 9-12 use a 2-dimensional array to hold an array of characters which represent a maze. The method of construction of the maze and the functional definitions of all the associated words are given in the glossary, Listings 13 and 14. Executing TEST will produce the maze shown in Fig. 3. Change 1 RAND to 0 RAND and TEST will produce a different maze each time.

Listings 16-18 use the maze produced by TEST to show how, from any position within the maze, an exit can be found (providing one exists). The method is recursive, ie the word SEEKEXIT actually contains a call of itself and this necessitates, in Abersoft's FORTH, the use of the word PGSMUDGE, as explained in the glossary. Again the method used to find the exit and the functional definitions of the associated words are contained in the glossary, Listings 19 and 20.

If you execute TEST (with 1 RAND in the definition as shown), followed by FINDEXIT, then the output should be as shown in Fig. 3.

I hope this month's article has whetted your appetite for a little more FORTH, as next month I'll be explaining how you can communicate with your program (ie input/output), how FORTH handles the RAM disc, and how you can combine this with what we know of defining words to manipulate and store text within the dictionary.

```
( LISTING 6 - DEFINING WORDS FOR 'MONSTER ARRAY')
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
0 CONSTANT MNAME   1 CONSTANT MSTRENGTH
2 CONSTANT MPSI    3 CONSTANT MCODE
: MARRAY <BUILDS
   19 0 DO
    CLS ." TYPE IN NAME OF MONSTER" CR
    QUERY 34 ( ASCII " ) TEXT PAD DUP 15 + SWAP
    DO I C@ C, LOOP
    CR ." TYPE IN MONSTER'S STRENGTH" CR QUERY INTERPRET C,
    CR ." TYPE IN MONSTER'S PSI STRENGTH" CR QUERY INTERPRET C,
    CR ." TYPE IN CODE FOR SCENE SELECTION" CR KEY DUP EMIT C,
  LOOP
  DOES>   ( ELEMENT,FIELD,BASE ADDRESS)
   OVER DUP MNAME = OVER MSTRENGTH = OR
   OVER MPSI = OR SWAP MCODE = OR NOT -->
```

Listing 6

```
( LISTING 7 - MONSTER ARRAY CONTINUED)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)

   IF CR ." ERROR! INVALID FIELD SUPPLIED" CR SP! QUIT THEN
   ( CALCULATE OFFSET FOR CHOSEN FIELD )
   ( ELEMENT,FIELD,BASE ADDR.)
   OVER ( E,F,BA,F)
   CASE MSTRENGTH OF 15 + ENDOF
        MPSI      OF 16 + ENDOF
        MCODE     OF 17 + ENDOF
   ENDCASE  ( E,F,OFFSET BASE ADDR.)
  ROT ( F,OBA,E) DUP 18 > OVER 0< OR
  IF CR ." ERROR! OUT OF BOUNDS IN MARRAY" SP! QUIT THEN
  18 * + ( F,ELEMENT ADDR)
  SWAP MNAME = IF COUNT ELSE C@ THEN
;
```

Listing 7

```
( LISTING 8 - RANDOM NUMBER GENERATOR)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)

0 VARIABLE SEED

: SEEDON ( -NEXT VALUE OF SEED)
  SEED @ 75 U* 75 0 D+ OVER OVER U< - - 1 - DUP SEED ! ;

: RAND ( VALUE FOR SEED-)
  DUP 0= IF 23672 C@ SWAP DROP THEN SEED ! ;

: RND ( N-PSEUDO RANDOM NUMBER BETWEEN 0 AND N-1 )
  SEEDON U* SWAP DROP ;


2 LOAD 3 LOAD 4 LOAD 5 LOAD 8 LOAD 9 LOAD
```

Listing 8

```
( LISTING 9 - WORDS FOR MAZE DRAWING)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
: VAR VARIABLE ;    : CON CONSTANT ;   ( MAKES TYPING EASIER )
31 VAR SCREEN-WIDTH   20 VAR SCREEN-HEIGHT   2 VAR WALLSIZE
0  VAR MAXROW         0  VAR MAXCOL          0 VAR LINES
: SETPARAMS ( -)
  WALLSIZE @ SCREEN-WIDTH @ OVER / * MAXCOL !
  WALLSIZE @ SCREEN-HEIGHT @ OVER / * MAXROW !
  MAXROW @ MAXCOL @ * WALLSIZE @ DUP * / 100 * 127 / LINES ! ;
72 ( ASCII H) CONSTANT HEDGE   46 ( ASCII .) CONSTANT WALKWAY
0 VAR ROW-START   0 VAR COL-START   0 VAR ROW-DIR
0 VAR COL-DIR     0 VAR END-ROW     0 VAR END-COL
SETPARAMS  ( SETS VALUES FOR MAXROW & MAXCOL)
MAXROW @ MAXCOL @ 2-DCHECK MAZE    ( SETS UP ARRAY SPACE)
: CPLOT ( ASCII CODE,ROW,COL)
  OVER OVER AT ROT DUP EMIT ROT ROT MAZE ! ;
```

Listing 9

```
( LISTING 10 - MAZE DRAWING CONTINUED)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
: CPOINT ( ROW,COL-ASCII CODE) MAZE @ ;
: PICKSTART WALLSIZE @ MAXCOL @ OVER / 1+ RND * COL-START !
     WALLSIZE @ MAXROW @ OVER / 1+ RND * ROW-START ! ;
: PICKDIRECTION 4 RND 1+ CASE
  1 OF  1 COL-DIR !  0 ROW-DIR ! ENDOF
  2 OF -1 COL-DIR !  0 ROW-DIR ! ENDOF
  3 OF  0 COL-DIR !  1 ROW-DIR ! ENDOF
  4 OF  0 COL-DIR ! -1 ROW-DIR ! ENDOF
  ENDCASE ;
: FINDEND WALLSIZE @ COL-DIR @ * COL-START @ + END-COL !
          WALLSIZE @ ROW-DIR @ * ROW-START @ + END-ROW ! ;
: CHECKEND ( -FLAG) END-COL @ 0<
   MAXCOL @ END-COL @ < OR END-ROW @ 0< OR MAXROW @ END-ROW @ <
   OR IF 1 ELSE END-ROW @ END-COL @ CPOINT HEDGE = THEN ;
```

Listing 10

```
( LISTING 11 - MAZE DRAWING CONTINUED)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
: PLOTLINE HEDGE ROW-START @ COL-START @ CPLOT WALLSIZE @  0 DO
  ROW-START @ ROW-DIR @ + ROW-START !
  COL-START @ COL-DIR @ + COL-START !
  HEDGE ROW-START @ COL-START @ CPLOT
  LOOP ;

: TRYLINES ( FLAG-FLAG)
  BEGIN DUP 0= IF PICKSTART THEN DROP 0 5 0
  DO PICKDIRECTION  FINDEND CHECKEND 0=
   IF PLOTLINE DROP 1 LEAVE THEN
  LOOP
  INKEY 32 = IF DROP 2 THEN DUP
  UNTIL ;
```

Listing 11

```
( LISTING 12 - MAZE DRAWING CONTINUED)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
: DRAWMAZE
  0 LINES @ 0 DO TRYLINES DUP 2 = IF LEAVE THEN LOOP DROP ;
: DRAWBORDER
  MAXROW @ 1+ 0 DO HEDGE I 0 CPLOT HEDGE I MAXCOL @ CPLOT LOOP
  MAXCOL @     1 DO HEDGE 0 I CPLOT HEDGE  MAXROW @ I CPLOT LOOP
  MAXCOL @ 1 DO MAXROW @ 1 DO WALKWAY I J CPLOT LOOP LOOP ;
: DISPLAYMAZE
  CLS MAXROW @ 1+ 0 DO MAXCOL @ 1+ 0 DO
  J I MAZE @ EMIT LOOP CR LOOP ;
: TEST CLS 1 RAND DRAWBORDER DRAWMAZE 20 0 AT ;
```

Listing 12

```
( LISTING 13 - MAZE DRAWING EXPLANATION)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
To draw a random pattern maze within a space limited by screen
height and screen width and where the gap between each parallel
wall shall be equal to the value stored in WALSIZE less one.
 First scale the boundaries of the maze so that there is always
the same space between adjacent parallel walls. The maze is
going to be drawn by plotting a series of small walls of length
WALSIZE +1. Calculate roughly how many of these small walls are
needed to fill the maze and store the number in LINES.
 The maze is made up of HEDGES letter H and WALKWAYS character
To draw the maze, draw a border and repeatedly for the number
of LINES  1) Pick a random starting point
          2) pick a random direction
     3) caculate the end point of the short wall
4) if it is not a hedge or outside the array then plot the wall.
```

Listing 13

```
( LISTING 14 - MAZE DRAWING: GLOSSARY OF WORDS)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
 NAME       STACK EFFECTS       DESCRIPTION
CPLOT (ascii code,row,col-) plots the given charcter both in the
      the array MAZE and on the screen.
CPOINT (row,col-ascii code) returns the character stored in the
        array MAZE at point row,col.
PICKSTART (-) selects a random starting point for new wall. Puts
            position in ROW-START & COL-START.
PICKDIRECTION (-) selects a random direction for wall. Puts
               values in ROW-DIR & COL-DIR
FINDEND (-) calculates end-point of possible new wall. Returns
         position in END-ROW & END-COL.
CHECKEND (-flag) Returns a vaue 1 on the stack if the end point
          is out of the array or is a HEDGE.
```

Listing 14

```
( LISTING 14 CONT. - MAZE DRAWING: GLOSSARY OF WORDS CONT.)
PLOTLNE (-) draws a wall WALLSIZE +1 long from
         ROW-START,COL-START to ROW-END,COL-END.
TRYLINES (flag-flag) if the flag on the stack is zero then
a new starting point is calculated for the next wall. Otherwise
the next wall will start from the end of the last wall plotted.
Five attempts are then made to draw a new wall. If it is not
possible then a new starting point is chosen.
 This routine can be interrupted by pressing the SPACE key.
 This will end the drawing of the maze.
 A flag is left on the stack indicating 1,successfull drawing
 of wall. 2,abandon drawing of maze.
DRAWMAZE (-) controls the drawing of the number of walls.
DRAWBORDER (-) draws a border for the maze.
DISPLAYMAZE (-) draws array containing maze on the screen.
TEST (-) will always produce the same maze.(Change 1 to alter.)
```

Listing 14 (continued)

```
( LISTING 16 - LEAVE MAZE ROUTINES)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
: UNFILL
  MAXROW @ 1 DO MAXCOL @ 1 DO J I CPOINT HEDGE =
   IF ( DO NOTHING) ELSE WALKWAY J I MAZE ! THEN
  LOOP LOOP ;
: MAKEEXIT WALKWAY MAXROW @ 2 - MAXCOL @ MAZE !
 WALKWAY  MAXROW @ 1 -  MAXCOL @  MAZE ! ;
0 VAR EASTLIMIT     0 VAR SOUTHLIMIT
0 CON NORTHLIMIT    0 CON WESTLIMIT
0 VAR EXITFOUND
111 ( ASCII o) CONSTANT FOOTSTEP
 42 ( ASCII *) CONSTANT PATHMARK
: PGSMUDGE SMUDGE ; IMMEDIATE
```

Listing 16

```
( LISING 17 - LEAVE MAZE ROUTINES CONT.)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
: SETLIMITS MAXCOL @ EASTLIMIT ! MAXROW @ SOUTHLIMIT ! ;
SETLIMITS
: SEEKEXIT ( LAT,LON-) PGSMUDGE R> ROT ROT
  >R >R I NORTHLIMIT    = I SOUTHLIMIT @ = OR
  I' WESTLIMIT = OR I' EASTLIMIT @ = OR R> R> ROT
  IF 1 EXITFOUND !
   ELSE
   OVER OVER FOOTSTEP ROT ROT CPLOT
   ( TRY EAST)
   OVER OVER 1+ CPOINT WALKWAY =
    IF OVER OVER 1+ SEEKEXIT THEN
   EXITFOUND @ NOT
   IF ( TRY SOUTH)
   OVER 1+ OVER CPOINT WALKWAY = --> ( COMPILE NEXT SCREEN)
```

Listing 17

```
( LISING 17 - LEAVE MAZE ROUTINES CONT.)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
    IF OVER 1+ OVER SEEKEXIT THEN
   THEN EXITFOUND @ NOT
   IF ( TRY WEST)
   OVER OVER 1 - CPOINT WALKWAY =
    IF OVER OVER 1 - SEEKEXIT THEN
    THEN EXITFOUND @ NOT
   IF ( TRY NORTH)
   OVER 1 - OVER CPOINT WALKWAY =
    IF OVER 1 - OVER SEEKEXIT THEN
   THEN
  THEN EXITFOUND @ IF PATHMARK ROT ROT CPLOT
  ELSE DROP DROP THEN >R PGSMUDGE ;
: LEAVEMAZE ( -) UNFILL MAKEEXIT 0 EXITFOUND ! DISPLAYMAZE
  1 1 SEEKEXIT 20 0 AT EXITFOUND @ 0= IF ." NO WAY OUT! "THEN ;
```

Listing 17 (continued)

```
( LISTING 18 - EXPLANATION OF METHOD TO LEAVE MAZE)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
Assume the maze is a rectangular enclosure divided into squares
each square being either covered by a hedge or not. The
perimeter squares are all hedge covered except for one or more
exits. You are released somewhere inside the maze and you have
to find your way to an exit. You may move from square to square
in any direction except diagonally, but you cannot cross a
hedge. The maze is represented in the two dimensional array
MAZE. Letter 'H' represents a hedge and character '.' a pathway.
( To find a path from square to an exit,a possible solution:-)
IF square S is on the perimeter THEN exit from maze
ELSE try heading East
 IF no exitfound yet THEN try heading South
 IF no exitfound yet THEN try heading West
 IF no exitfound yet THEN try heading North    ( end)
```

Listing 18

```
( LISTING 19 - EXPLANATION OF METHOD TO LEAVE MAZE CONT.)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
We can further refine 'try heading East' ( the others will be
similar)
( Try heading east)
IF S's eastern neighbouring square is a pathway THEN
   find a path from S's eastern neighbouring square to an exit.
But 'find a path from...etc.' is the same as the original
problem. So we can use a recursive word ( i.e. one that contains
a call of itself) that expects on the stack the coordinates of
its starting square.
The proceedure marks each square it visits with a 'footstep'
so that it doesn't go round in circles. The proceedure when
it has found an exit marks each square which lies on the path
with an asterisk (*). The final picture of the maze will show
the path and any blind alleys which were followed ( listing 21).
```

Listing 19

```
( LISTING 20 - GLOSSARY OF WORDS FOR LEAVE MAZE ROUTINES)
( 48K SPECTRUM   ABERSOFT fig-FORTH 1.1A)
UNFILL (-) Empties the maze of pathmarks and footsteps. Gives
            you a 'clean' maze withouthaving to re-draw it.
MAKEEXIT (-) forms an exit in the bottom right hand corner.
PGSMUDGE (-) Something I've not explained yet but this word
              has to be used to form a RECURSIVE definition.
SETLIMITS (-) Initialises the variables that SEEKEXIT uses to
               determine if the exit of the maze has been found.
SEEKEXIT ( row,col-) uses the row & column provided on the
stack as the starting point from which to try and find an exit.
The use of the words R> & >R will be explained in a further
article. BE CAREFUL to type an equal number of R>'s and >R's
as shown in the listing!
LEAVEMAZE (-) will give an output as listing 21 if it uses
the maze produced by TEST.
```

Listing 20

THE NEW MPF1 PLUS...

...THE LOWEST COST
Z80 SINGLE BOARD COMPUTER
AVAILABLE WITH ALL THESE FEATURES!

BUILT-IN ASSEMBLER

The MPF1 PLUS incorporates the Z80 – the most widely used 8-bit microprocessor in the world, to form a Single Board Computer (SBC). Packed in a plastic bookcase together with three comprehensive manuals and power supply (to BS3651 standard), the MPF1 PLUS is a microprocessor learning tool for every application.

Teaching you in a step-by-step method the MPF1 PLUS helps the user fully understand the Software and Hardware of a microprocessor easily and conveniently – as opposed to micro-computers that aim to teach high-level languages instead of microprocessor systems fundamentals.

Not only is the MPF1 PLUS a teaching tool but with the available accessories it can also be used as a low-cost development tool or simply for OEMs.

FLIGHT
Electronics Ltd.

Quayside Rd, Southampton, Hants SO2 4AD. Telex 477793. Tel. (0703) 34003/27721.

Micro-Professor is a trade mark of Multitech Industrial Corporation. Z80 is a trade mark of Zilog Inc.

THE MPF1 PLUS

Just look at the specification:-

Technical Specification

CPU: Z80A – 158 instructions
Software:
- Z80/8080/8085 machine code
- Z80 Assembler, line and 2 pass.
- 8K BASIC interpreter (Extra)
- 8K FORTH (Extra)

ROM: 8K Monitor (full listing and comments)
RAM: 4K CMOS (2 x 6116)
Input/Ouput: 48 system I/O lines
Speaker: 2.25" coned linear
Display: 20 character 14 segment green phosphorescent
Expansion:
- Socket for 8K ROM
- Cassette interface
- Connectors 40 way, complete CPU bus

Keyboard: 49 key. Full "QWERTY" real movement good tactile feedback
Batteries: 4 x U11 for memory back-up (batteries not included)
Serial Interface: 165 baud for read/write via audio cassette

Manuals

1. User's Manual. 8 chapters.
   1. Over view and Installation.
   2. Specification (hardware and software). 3. Description of Operation. 4. Operating the MPF-1 Plus. 5. 44 Useful Sub-Routines.
   6. The Text Editor.
   7. Assembler and Disassembler.
   8. System Hardware Configuration.
2. Experiment Manual. 16 experiments.
3. Monitor Program Source Listing with full commenting.
4. Also available the MPF-1 Plus Student Work Book (self-learning text).

Accessories

- **PRT-MPF-1P:** 20 character printer. Ready to plug in. Memory dump.
- **EPB-MPF-1P:** Copy/list/verify 1K/2K/4K/8K ROMS. Ready to plug in.
- **SSB-MPF-1P:** Speech Synthesizer. Inc. 20 words and clock program. 1200 words available.
- **SGB-MPF-1P:** Sound Synthesizer Board.
- **I/O – MPF-1P:** Input/output board

Yes! I now realise that I need an MPF1 PLUS and that it is the lowest cost Z80 SBC available with all these features.

I enclose £165.00 (£140.00 + £21 VAT plus £4 carriage). Overseas P.O.A.

Cheques payable to FLIGHT ELECTRONICS LTD.

Please debit my Barclaycard/Access ☐ ☐

Account No. ____________

An invoice will automatically be sent.

Name ____________

Address ____________

____

____

Signature ____________

Date ____________

ETI

Unlock Your Imagination
SCOPE
Computer Graphics Language
★ A VERY POWERFUL, TRUE MULTITASK LANGUAGE
★ AS FAST AS MACHINE CODE
★ SIMPLER THAN BASIC
Write Machine Code in a fraction of the time currently required
SCOPE is available from most good quality dealers and selected branches of WHSMITH
For details of how to get your games published commercially contact ISP Marketing
PRICE
£11.95
TECHNICAL DETAILS
SCOPE is a fully structured multitask language specifically for writing graphics, animation and sound. Being fully compatible with BASIC it is ideal for writing both arcade and adventure style games. Additionally, with SCOPE present in high memory it can be used as an assembler with SCOPE words as plain language mnemonics. Therefore no knowledge of machine code is required. The language is extremely easy to comprehend and is very powerful indeed providing many features not available from BASIC.
SCOPE HAS TO BE USED TO BE BELIEVED
48K SPECTRUM
Unlock Your Imagination
SCOPE
CLIP THE COUPON NOW AND UNLOCK YOUR IMAGINATION
Post to: ISP Marketing Ltd. Crown House, 38b High Street, Godalming, Surrey
Tel: Godalming(04868) 24151 Please rush me ..... copies of the SCOPE computer graphics language tape and instruction manual at £11.95 (p&p free).
(Cheques payable to ISP Marketing Ltd)
Block capitals please
Name
Address
Dealer Enquiries Welcome
D

M. D. Buxton

NON-RANDOM RANDOM NUMBERS

The whole point of random numbers is that they are unpredictable, so a program that makes use of them can be very difficult to debug. This article offers a simple solution using a resource at hand in every computer.

Have you ever needed a set of random numbers for debugging, testing or other purposes? Perhaps you have used a simple subroutine such as this which will generate 100 random numbers in the range 1-10:

```
10010 FOR I=1 TO 100
10020 R=RND(10)
10030 PRINT R
10040 NEXT I
10050 RETURN
```

However, the random numbers will not be the same every time that subroutine is used. One way of getting round this is to put the numbers in an array and extract them as required:

```
10 DIM R(100)
.
.
.
10010 FOR I=1 TO 100
10020 R=RND(10)
10030 R(I)=R
10040 NEXT I
10050 RETURN
```

Another simple subroutine in the same program can be used to extract these random numbers so that they reappear in the same order:

```
20010 FOR I=1 TO 100
20020 PRINT R(I)
20030 NEXT I
20040 RETURN
```

Unfortunately, the sequence of random numbers will stay unchanged only during one run. The advantage over the techniques used in the first subroutine is that the array can be examined as many times as necessary during the run. A disadvantage is that large arrays need a lot of memory. Where practicable, integer arrays should be used in this sort of application.

How about using READ and DATA statements? The data read in would be the same for every run! Let's see:

```
10 DIM R(10)
20 DATA 1,9,4,4,3,7,8,5,6,2
30 FOR I=1 TO 10
40 READ R
50 R(I)=R
60 NEXT I
.
.
.
20010 FOR I=1 TO 10
20020 PRINT R(I)
20030 NEXT I
20040 RETURN
```

Well, a program based on this is not so bad for only 10 'random' numbers but what about a 100 or a 1000? For some purposes, it may not be necessary to hold the numbers in an array (economising on memory) but rather use READ and RESTORE. There is still the problem of slaving over a hot keyboard, keying in that innumerable 'random' data.

NON-RANDOM RANDOMS

There is an easier way, if you do not mind the fact that the numbers generated are not genuinely random and do appear to have a bias in their distribution. Have you PEEKed your computer's ROM to find a mass of seemingly random numbers in the range 0 to 255? The Level II ROM of the TRS-80 Model I occupies addresses 0 to 12287, so there are plenty of these 'random' numbers available. PEEKing addresses other than those in the ROM reveals more numbers but these are not always reproducible. The contents of the addresses in the ROM do not change in the course of running a program unless there is a catastrophe! Such an event is likely to discourage to computer from functioning at all.

Now that we have 'random' numbers, what to do with them? Can they be manipulated to produce a set of numbers within a certain range? In short, yes they can. One technique is to PEEK consecutive addresses and accept the contents if they lie within a certain range.

Another way is to scale the numbers obtained. This sort of subroutine will produce 'random' integers in the range 1-10:

```
10010 FOR I=1 TO 100
10020 R=INT(PEEK(I)/25)
10030 PRINT R
10040 NEXT I
10050 RETURN
```

This subroutine will always produce the same 100 'random' numbers. There is no need to use an array to keep them safely; they are readily accessible.

This sort of technique may not suit your requirements when random numbers considerably larger than 255 are needed. One way of producing 'random' numbers up to 65535 is to add the contents of one address to 256 times the contents of another address:

```
10010 FOR I=1 TO 100
10020 R=PEEK(I) + 256*PEEK(I+1)
10030 PRINT R
10040 NEXT I
10050 RETURN
```

Another technique? Not a few programs involve branching if a condition is met. Perhaps something along these lines:

```
10 FOR I=1 TO 1000
20 IF PEEK(I)>204 THEN PRINT
   "HIGH"
30 NEXT I
```

Roughly 23% of the contents of the addresses are greater than 204, causing "HIGH" to be printed about 230 times. This information can be used to create random mazes at higher speeds than those created in this sort of way:

```
10 RANDOM
20 CLS
30 R=RND(11200)
40 FOR I=0 TO 1023
50 IF PEEK(I+R)>204 THEN POKE
   15360+I,191
60 NEXT I
70 PRINT@0,"READY";
80 GOTO 80
```

The program generates random numbers between 1 and 100. If the number is less than 24 then a graphic block is POKEd into the video RAM which starts at address 15360 and occupies 1024 bytes. This is done for all the 1024 addresses, resulting in a random maze of about 235 graphic blocks. The program occupies 135 bytes and takes about 20 seconds to produce the maze. Note that the maze is different every time that the program is run. Now try this program:

```
10 CLS
20 FOR I=0 TO 1023
30 IF PEEK(I)>204 THEN POKE
   15360+I,191
40 NEXT I
50 PRINT@0,"READY";
60 GOTO 60
```

This program occupies 113 bytes and takes about 10 seconds to produce the maze which is the same every time that the program is run.

Now suppose that you like the speed advantage but don't want the reproducibility of the maze? Simply start PEEKing from a different address, for example this program which takes up 126 bytes:

```
10 RANDOM
20 CLS
30 FOR I=0 TO 1023
40 R=RND(100)
50 IF R<24 THEN POKE 15360+I,191
60 NEXT I
70 PRINT@0,"READY";
80 GOTO 80
```

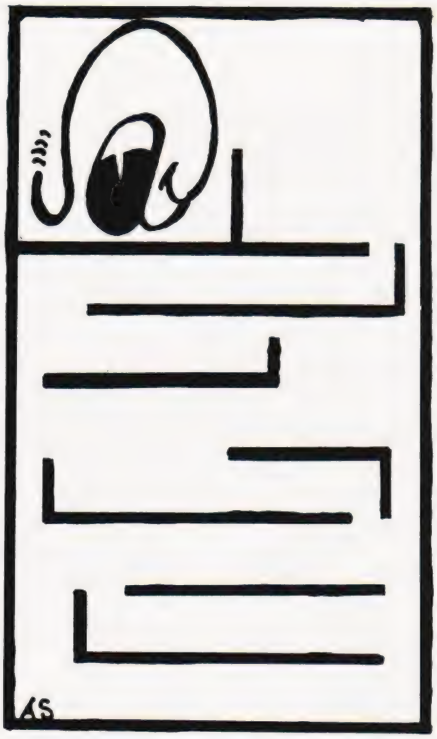
This program still occupies less memory than that in Listing 8 and takes about 13 seconds to produce a maze.

I have not exhausted all possibilities, such as the use of random numbers to produce random letters, but such delights I leave to you. However, as an example of a game which utilises reproducible random numbers, there is . . . ATMAN!

ATMAN-IA

The computer creates a maze through which a solitary figure is directed home by use of the numeric keypad. A bulky slow-witted ATMAN will probably try to stop its progress by jumping on it. Should the ATMAN succeed, you will be given another chance to try. In the event that you succeed, you will be given the chance to try against an increased number of ATMEN. The game continues until you give up or eventually defeat the maximum of 20 ATMEN.

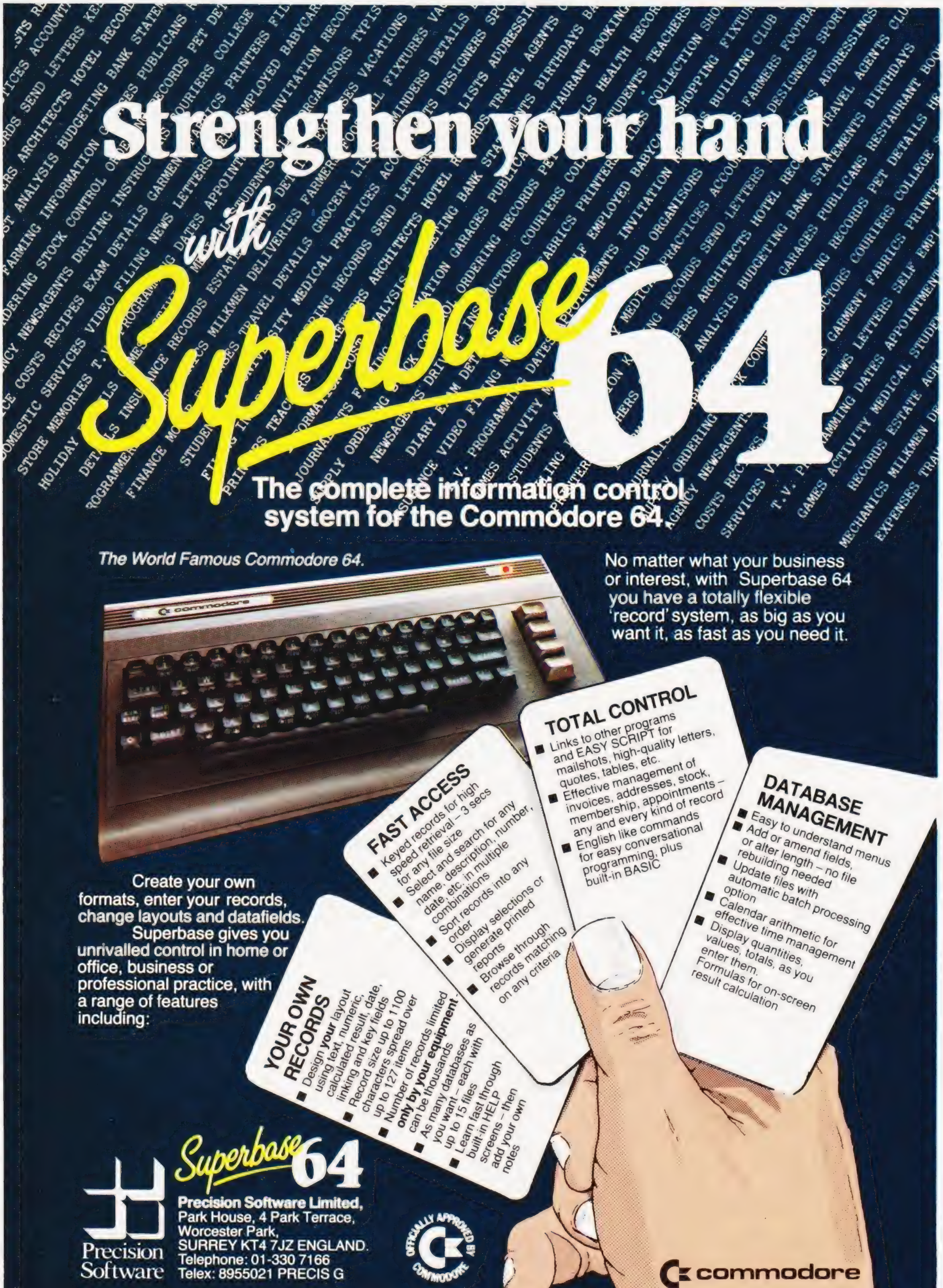
As it stands, the program occupies over 6K of RAM. By omitting REM statements and spaces it can be reduced to less than 4K. The copious REMs should make the program self-explanatory.



Listing 1. The ATMAN program.

```
10 REM ** ATMAN BY M.D. BUXTON 1983
20 REM ** INITIALIZE VARIABLES
30 DEFINT A-Z
40 REM ** START OF VIDEO RAM
50 C=15360
60 REM ** @ SYMBOL AND SCREEN WIDTH
70 C1=64
80 REM ** INCREMENT OF MOVEMENT
90 C2=1
100 REM ** GRAPHICS BLOCK
110 C3=191
120 RANDOM
130 REM ** ATMAN ARRAY
140 DIM R(20,3)
150 REM ** NUMBER OF ATMEN
160 NR=1
170 REM ** SET UP MAZE AND INSTRUCTIONS IF REQUIRED
180 GOTO 1970
190 REM ** BUILD UP HOME REGION
200 PRINT@31,"[4 SPC]";
210 PRINT@95,"[4 SPC]";
220 PRINT@159,"[4 SPC]";
230 REM ** THIS FORCES AN UPPER CASE 'X'
240 POKE 15436,88
250 REM ** BORDERS FOR MESSAGES
260 FOR Z=C2 TO 15
270 POKE C+C1+Z,C3
290 NEXT Z
300 REM ** BUILD UP PLAYER'S START
310 REM ** PLAYER IS 'O'
320 PRINT@860,"[5 SPC]";
330 PRINT@924,"[2 SPC]O[2 SPC]";
340 PRINT@988,"[5 SPC]";
350 REM ** SET KEYBOARD MOVEMENT NUMBER TO ZERO
360 K=0
370 IF NR=1 THEN PRINT@45,"****************";
380 IF NR>1 THEN PRINT@0,"*";NR;"* ATMEN *";
390 GOTO 1780
400 REM ** PLAYER PROMPT
410 PRINT@45,"* YOUR * MOVE *";
420 REM ** HAS KEY BEEN PRESSED TO ALTER DIRECTION?
430 Z$=INKEY$
440 IF Z$="" THEN 510
450 REM ** KEYBOARD DIRECTION
460 REM ** THIS PREVENTS 'FREEZING' BY PRESSING 0,5
    OR A LETTER
470 L=VAL(Z$)
480 IF L=0 OR L=5 THEN 430
490 K=VAL(Z$)
500 REM ** AFTER SETTING UP, K SHOULD BE ZERO
510 IF K=0 OR K=5 THEN 430
520 REM ** NEW POSITION OF PLAYER
530 ON K GOTO 570,600,630,660,690,700,730,760,790
540 REM ** NEW CO-ORDINATES CALCULATED
550 REM ** TX IS NEW VALUE OF X
560 REM ** TY IS NEW VALUE OF Y
570 TX=X-C2
580 TY=Y+C2
590 GOTO 830
600 TX=X
610 TY=Y+C2
620 GOTO 830
630 TX=X+C2
640 TY=Y+C2
650 GOTO 830
660 TX=X-C2
670 TY=Y
680 GOTO 830
690 GOTO 1540
700 TX=X+C2
710 TY=Y
720 GOTO 830
730 TX=X-C2
740 TY=Y-C2
750 GOTO 830
760 TX=X
770 TY=Y-C2
780 GOTO 830
790 TX=X+C2
800 TY=Y-C2
810 GOTO 830
820 REM ** KEEP ON SCREEN
830 IF TX>63 THEN TX=63 ELSE IF TX<0 THEN TX=0
840 REM ** PREVENTING WRAP-AROUND
850 IF TY>16 THEN TY=16 ELSE IF TY<0 THEN TY=0
860 REM ** TP IS TO BE PEEKED (NEW LOCATION OF
    PLAYER
870 TP=C+TY*C1+TX
880 REM ** KEEP ON SCREEN
890 IF TP>16383 THEN 1110
900 REM ** CURRENT LOCATION OF PLAYER
910 PP=C+Y*C1+X
920 REM ** JUMPING TO WHAT?
930 TL=PEEK(TP)
940 REM ** SPACE
950 IF TL=32 THEN 1030
960 REM ** ATMAN
970 IF TL =C1 THEN 1540
980 REM ** HOME AND WIN
990 IF TL=88 THEN 1030
1000 REM ** WALL OR OTHER - NO MOVE
1010 GOTO 1110
1020 REM ** POKE PLAYER INTO NEW POSITION
1030 POKE TP,79
1040 REM ** POKE BLANK INTO PLAYER'S OLD POSITION
1050 POKE PP,32
```

```
1060 REM ** UPDATE
1070 X=TX
1080 Y=TY
1090 IF TL=88 THEN 1610
1100 REM ** REPLACE PLAYER PROMPT
1110 PRINT@45," ATMAN * MOVES ";
1120 FOR I=C2 TO NR
1130 REM ** CAN THE ATMAN MOVE? DELETE THE REM FROM
     LINE 1150 TO ALTER THE CHANCE OF AN ATMAN MOVING
     FROM CERTAIN TO 1 IN 3
1140 R(I,3)=-C2
1150 REM ** IF RND(10)>3 THEN R(I,3)=0
1160 NEXT I
1170 FOR I=C2 TO NR
1180 REM ** ASSIGN VALUES TO TEMPORARY VARIABLES
1190 REM ** BX IS HORIZONTAL CO-ORDINATE
1200 BX=R(I,1)
1210 REM ** BY IS VERTICAL CO-ORDINATE
1220 BY=R(I,2)
1230 REM ** MOVE FLAG NOT SET?
1240 IF R(I,3) THEN 1520
1250 REM ** HORIZONTAL AND VERTICAL DISTANCES
1260 REM ** DX IS HORIZONTAL DISTANCE FROM ITH ATMAN
     TO PLAYER
1270 DX=ABS(R(I,1)-X)
1280 REM ** DY IS VERTICAL DISTANCE FROM ITH ATMAN
     TO PLAYER
1290 DY=ABS(R(I,2)-Y)
1300 REM ** WHICH DISTANCE IS GREATER?
1310 IF DX<DY THEN 1360
1320 REM ** ADJUST HORIZONTAL DISTANCE
1330 IF R(I,1)>X THEN BX=R(I,1)-C2 ELSE BX=R(I,1)+C2
1340 GOTO 1380
1350 REM ** ADHUST VERTICAL DISTANCE
1360 IF R(I,2)>Y THEN BY=R(I,2)-C2 ELSE BY=R(I,2)+C2
1370 REM ** BP IS PROVISIONAL LOCATION OF ITH ATMAN
1380 BP=C+BY*C1+BX
1390 REM ** KEEP ON SCREEN
1400 IF BP>16383 OR BP<C THEN 1520
1410 BC=PEEK(BP)
1420 REM ** STUCK?
1430 IF BC=32 OR BC=79 THEN 1450 ELSE 1520
1440 REM ** MOVE ATMAN
1450 POKE BP,C1
1460 POKE (C+R(I,1)+R(I,2)*C1),32
1470 REM ** COLLISION
1480 IF BC=79 THEN 1540
1490 REM ** UPDATE
1500 R(I,1)=BX
1510 R(I,2)=BY
1520 NEXT I
1530 GOTO 410
1540 PRINT@(X+C1*Y),"SPLAT!";
1560 GOSUB 2530
1570 IF Z$<>CHR$(13) THEN 1560
1580 CLS
1590 PRINT "YOU HAVE FAILED."
1600 GOTO 1730
1610 PRINT@0,"  SUCCESS!!!  ";
1620 REM ** INCREMENT NUMBER OF ATMEN
1630 NR=NR+C2
1640 PRINT@512,"PRESS ENTER TO CONTINUE.";
1650 GOSUB 2530
1660 IF Z$<>CHR$(13) THEN 1650
1670 IF NR<21 THEN 1710
1680 PRINT@512,"YOU HAVE DEFEATED ALL OF THE ATMEN!
     WOULD YOU LIKE TO START AGAIN? Y/N "
1690 GOSUB 2530
1700 IF Z$="Y" THEN RUN ELSE END
1710 PRINT@512,"WOULD YOU LIKE TO TRY AGAIN WITH ONE
     MORE ATMAN? Y/N "
1720 GOTO 1740
1730 PRINT@512,"WOULD YOU LIKE TO TRY AGAIN WITH THE
SAME NUMBER OF ATMEN? Y/N"
1740 GOSUB 2530
1750 IF Z$="N" THEN END ELSE 1970
1760 REM ** SET UP ATMAN ARRAYS
1770 REM ** 20 ATMEN
1780 FOR I=C2 TO  NR
1790 REM ** RANDOM COLUMN
1800 RX=RND(C1)-C2
1810 REM ** RANDOM ROW EXCEPT TOP
1820 RY=RND(15)
1830 REM ** LOCATION IN MEMORY
1840 RL=C+RX+C1*RY
1850 REM ** SPACE FOR ATMAN
1860 IF PEEK(RL)<>32 THEN 1800
1870 REM ** X CO-ORD
1880 R(I,1)=RX
1890 REM ** Y CO-ORD
1900 R(I,2)=RY
1910 REM ** FLAG FOR MOVEMENT
1920 R(I,3)=0
1930 NEXT I
1940 IF NR=1 THEN PRINT@0," * ONE * ATMAN ";
1950 REM ** PLAYER PROMPT
1960 GOTO 410
1970 CLS
1980 PRINT"ATMAN GAME"
1990 PRINT"@@@@@@@@@@"
2000 PRINT
2010 PRINT"WOULD YOU LIKE INSRUCTIONS? Y/N "
2020 GOSUB 2530
2030 IF Z$="N" THEN 2290
2040 CLS
2050 PRINT" I WILL PRINT A RANDOM MAZE COMPRISING
     SPACES
     WALLS            ";CHR$(C3);"
     ATMEN            @
     A HOME           X
     AND YOURSELF     O"
2060 REM ** THESE POKES FORCE THE CORRECT CHARACTERS
     IF THE LOWER CASE DRIVER PROGRAM OF SEPTEMBER
     '82 CT IS BEING USED
2070 POKE 15505,C3
2080 POKE 15633,88
2090 PRINT
2100 PRINT"USE THE NUMERIC KEYPAD TO DIRECT YOUR MOVE
MENT AND EVENTUALLY GET HOME."
2110 PRINT
2120 PRINT"7=UP AND LEFT     8=UP      9=UP AND RIGHT"
2130 PRINT
2140 PRINT"4=LEFT                     6=RIGHT"
2150 PRINT
2160 PRINT"1=DOWN AND LEFT    2=DOWN    3=DOWN AND RIGH
T"
2170 PRINT
2180 PRINT
2190 PRINT"USING ANY OTHER KEY WILL NOT WIN THE GAME
FOR YOU!"
2200 PRINT"PRESS ANY KEY TO CONTINUE."
2210 GOSUB 2530
2220 CLS
2230 PRINT"*HITTING THE KEYS RAPIDLY WILL NOT RESULT
IN FASTER MOVEMENT*YOU MAY ALTER DIRECTION WHEN YOU
HAVE SEEN THE * YOUR MOVE * MESSAGE APPEAR IN THE TOP
RIGHT HAND CORNER OF THE SCREEN."
2240 PRINT"FAT ATMEN WHICH CAN'T GET BETWEEN DIAGONAL
LY JOINING WALLS (UNLIKE YOU) WILL TRY TO JUMP ON YOU
! THEY ARE PERSISTENT BUT NOT VERY INTELLIGENT. THEY
ARE NOW HIDING IN THE WALLS.";
2250 PRINT" SOME OF THE MORE DEVIOUS ATMEN WILL NOT A
PPEAR IMMEDIATELY. SOME WILL WAIT, HOPING THAT AS YOU
PASS THEY WILL BE ABLE TO JUMP ON YOU IN A SURPRISE A
TTACK! LESS PATIENT OR EVEN LESS INTELLIGENT ATMEN WI
LL APPEAR FOR NO APPARENT REASON."
2260 PRINT"PRESS ANY KEY TO CONTINUE."
2280 GOSUB 2530
2290 CLS
2300 REM** CO-ORDS FOR PLAYER START
2310 X=30
2320 Y=14
2330 CLS
2340 PRINT"PRESS R FOR A RANDOM MAZE. PRESS ANY OTHER
KEY FOR AN UNCHANGING MAZE."
2350 GOSUB 2530
2360 IF Z$<>"R" THEN 2460
2370 CLS
2380 REM ** SELECT ROM AREA
2390 R=RND(11200)
2400 FOR P=R TO R+1023
2410 REM ** WALL OR SPACE
2420 IF PEEK(P)>220 THEN POKE (P-R+C),C3
2430 NEXT P
2440 REM ** RANDOM MAZE DONE
2450 GOTO 200
2460 CLS
2470 REM ** THIS IS TO CREATE AN UNCHANGING RANDOM
     MAZE
2480 FOR P=0 TO 1023
2490 IF PEEK(P)<25 THEN POKE C+P,C3
2500 NEXT P
2510 GOTO 200
2520 REM ** KEYBOARD SCAN ROUTINE
2530 Z$=INKEY$
2540 IF Z$="" THEN 2530 ELSE RETURN
2550 END
```



Strengthen your hand
with
Superbase 64
The complete information control system for the Commodore 64.
The World Famous Commodore 64.
No matter what your business or interest, with Superbase 64 you have a totally flexible 'record' system, as big as you want it, as fast as you need it.
Create your own formats, enter your records, change layouts and datafields. Superbase gives you unrivalled control in home or office, business or professional practice, with a range of features including:
YOUR OWN RECORDS
■ Design your layout using text, numeric, calculated result, date, linking and key fields
■ Record size up to 1100 characters spread over up to 127 items
■ Number of records limited only by your equipment
■ As many databases as you want – each with up to 15 files
■ Learn fast through built-in HELP screens – then add your own notes
FAST ACCESS
■ Keyed records for high speed retrieval – 3 secs for any file size
■ Select and search for any name, description, number, date, etc. in multiple combinations
■ Sort records into any order
■ Display selections or generate printed reports
■ Browse through records matching on any criteria
TOTAL CONTROL
■ Links to other programs and EASY SCRIPT for mailshots, high-quality letters, quotes, tables, etc.
■ Effective management of invoices, addresses, stock, membership, appointments – any and every kind of record
■ English like commands for easy conversational programming, plus built-in BASIC
DATABASE MANAGEMENT
■ Easy to understand menus
■ Add or amend fields, or alter length – no file rebuilding needed
■ Update files with automatic batch processing option
■ Calendar arithmetic for effective time management
■ Display quantities, values, totals, as you enter them.
Formulas for on-screen result calculation
Superbase 64
Precision Software
Precision Software Limited,
Park House, 4 Park Terrace,
Worcester Park,
SURREY KT4 7JZ ENGLAND.
Telephone: 01-330 7166
Telex: 8955021 PRECIS G
OFFICIALLY APPROVED BY COMMODORE
commodore

Henry Budgett

LOSTOCK SCREEN EDITOR

The Apple's archaic editing system is enough to give anyone the pip. Now the Lostock Screen Editor offers an improved performance for owners of the II, II+ and IIe.

One of the most fundamental utility programs built into any computer's operating system must surely be its screen editor. Without such a routine the user is forever condemned to retyping everything that has a mistake in it. While this may just about be tolerated on a single word it certainly causes immense frustration when an 80-character line of BASIC has to be keyed again.

There are three common types of screen editor; the line editor, the memory editor and the text editor. The first of these dates from the days when computers were rare and access was only available through a remote terminal such as a Teletype. The incorrect line could be recalled and edited with a series of commands; the Tandy series of computers still retains this system, as does the Dragon 32.

The memory editor first appeared on the Commodore PET and is regarded by some as the best form of screen editor you can have. To edit a line all you have to do is move the cursor to the incorrect portion, change it and press Return. However, it is possible to create phantom lines and once a change is made you cannot see the previous state of the line on the screen. I personally don't favour this kind of editor as it's all too easy to make serious mistakes, but it does have the advantage of speed and ease of use.

As a half-way house between the distinct clumsiness of a line editor and the speed of the memory editor there is the text editor. Possibly the best example currently available is the one supplied on the BBC Microcomputer/Electron. The idea is simple; you move a 'copy' cursor around with the direction keys and anything that you wish to add to the new line is actioned by the Copy Key. In this way a new line can be built up from parts of other lines as well as new input from the keyboard. There are no clumsy command sequences to remember, which makes it much simpler to use than a line editor, and the original version is retained on the screen for comparison.

ONE BAD APPLE

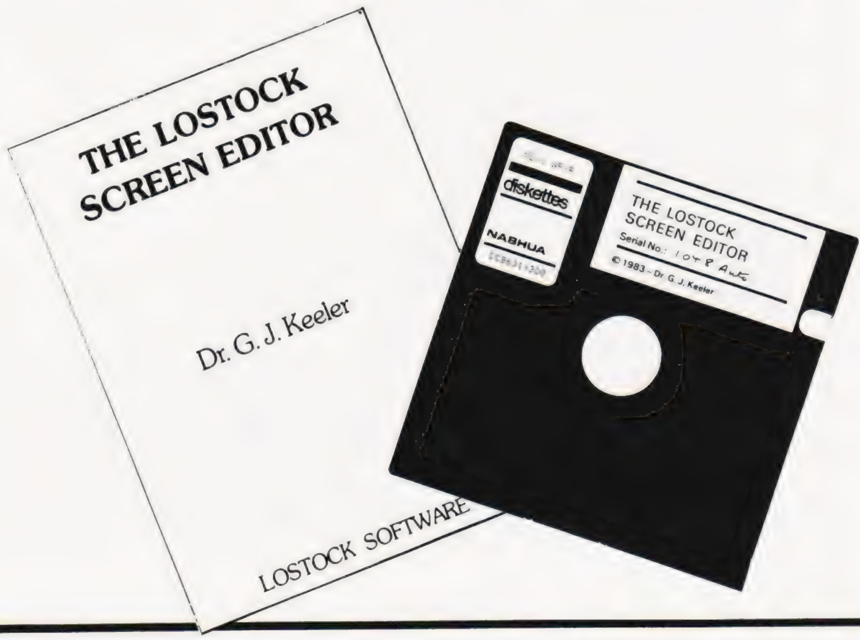
Of all the microcomputer editing functions which have been based around these three systems the one that stands out on its own is that of the Apple II. It is, quite simply, appalling! Despite the improvements when the Apple II+ was introduced, and the addition of a full set of cursor keys on the Apple IIe, it must still rank as the least useable editor ever.

To edit a line of program or even a mistyped command is slow and awkward. The original (Apple II) method required that the Control key be used in conjuction with the WZAS keys to give Up, Down, Left and Right movements of the cursor. The Apple II+ improved this slightly by offering the Escape Mode editor. Here the cursor movements are controlled by the IMJK keys once the Escape key had been pressed. In both cases the Right arrow key would copy the characters it passed over into the line buffer and the Left arrow would delete characters. Unfortunately, the characters you were putting into the buffer couldn't be seen and it was all too easy to miss letters or delete characters. To further compound the problem, the Applesoft interpreter re-formats the spacing of the lines displayed by LIST. While this makes the programs very legible it can make editing them a nightmare; text strings get split and umpteen spaces are added which must be stepped over or they end up in the new line.

Small wonder, then, that among the most popular utility programs for the Apple are decent line editors. One of the more recent offerings is the Lostock Screen Editor which is supplied as a HELLO program on disc together with a manual. For those readers unfamiliar with the Apple it is probably worth explaining the function of the HELLO program. Apple discs can be created as Masters or Slaves; the former carry all the DOS information while the latter do not. When an Apple is powered up with a Master disc in its drive it automatically loads a file called HELLO from the disc. As well as loading the DOS this can be a program in its own right, so the user can set up a menu or automatically load utility programs. In the case of the Lostock Screen Editor it loads and initialises the editor program.

In use the editor converts the Apple's primitive editing facility into something approaching the BBC Micro's text editor. To initialise the editing function, simply type Control-W and the normal Apple cursor is replaced by an underline symbol. This is the copy cursor and can be moved around the screen with the Control-WZAS keys.

Once the copy cursor is over the part of the line you wish to copy, simply press the Right arrow — the letters that it copies will appear on the input line. Deletion may be achieved with the Left arrow: characters are removed from the input line and not the copy cursor position.



Apple owners with this package can let their fingers do the walking more easily. . .

Full compatibility with the Apple II+ and Apple IIe is maintained so both Escape-IMJK and the arrow keys can be used too. In the Escape mode of editing, the copy cursor becomes an inverse E and the Space bar is used instead of the Right arrow to change from movement to copying.

While these functions go a long way to relieving the problems — now, at least, you can **see** what you've copied — they aren't the end of the Lostock's repertoire. If your mistake was in the first section of the line and you don't want to be bothered with trudging the cursor across the rest of the screen just press Control-E. This copies the rest of the screen line into the buffer and moves down to the beginning of the next screen line. Neat, but care must be taken to remember to press Return to enter the completed line — some wonderful things can happen if you forget!

The other major bugbear that the Lostock editor solves is that of the LIST formatting. Normally the experienced Apple user will do a quick POKE 33,33 before LISTing and squash the screen width. This dodge causes the formatting to be suppressed. Using the Lostock editor method is rather neater. If at any time during a line edit you want to tell the editor to ignore all the extra spaces just press Control-Q — when the cursor gets to the end of the screen line it jumps to the beginning of the text on the next screen line. This mode can be cancelled (sometimes there is a conflict between BASIC's spacing and the LIST format) by Control-P. The final touch provided is a double-keystroke LIST which is actioned by Control-L — well it's a 50% saving I suppose!

AN EDITOR'S HOME. . .

The Lostock editor lives in the section of Apple's memory between the DOS and the input buffers and this means that it's pretty safe from attack by your BASIC programs. Many commercial packages and other programming aids also seem to have this area pencilled in for their living space, so it's worth checking their compatibility with Lostock. One certain way of 'disconnecting' the editor is by the IN# command, and the manual recommends that IN#0 is used to effect a controlled disconnection. To get the editor back (who needs these transplants anyway!), all that is needed is a quick Reset or an & <Return> from Applesoft. Monitor users can get their functions back with Control-Y <Return>.

The main reason for wanting to disconnect the editor is that with an Apple IIe running an 80-column card, some of the Control functions clash. Examples of this are Control-W which scrolls the screen, Control-Z which clears the current line and Control-Q which reverts the display to 40-column! Perhaps it's not such a good idea to use it with an active 80 column card.

The Lostock editor is normally supplied on its own but for an extra £5 it comes with an extra programmers' aid, an auto line numberer. Whether my copy was sick or there is a genuine bug I'm not sure, but during the initialisation of the facility some odd things happen. To turn the numberer on one simply presses Control-A and a prompt appears asking for the first line number, the default being 10. However, if you press Return to either this prompt or the one asking for the increment, all sorts of garbage appears under the cursor. The facility worked perfectly otherwise but there's something not quite right with its manners.

As a visible means of support the Lostock editor is supplied with a slim 16-page A5 booklet. Mind you, with just five commands to remember for the editor, plus three for the auto numberer, the manual is sufficient. Indeed, the last page of text (the last three pages are blank) is a quick reference table and that's all I use these days. There appear to be no omissions or mistakes in the documentation and the known problems are outlined, although not covered in depth, so it serves its purpose.

Probably the only conclusion that I can draw from using the editor, and I've been doing so for several months, is that it seems to have become firmly attached to every one of my BASIC development discs! In terms of value for money it probably doesn't score very highly — £5 for the addition of an auto line numberer is a little steep — but then it has certainly saved both time and frustration during the preparation of two issues-worth of **Orchard Computing.** What it does show up, though, is that the Apple's screen editor is still one of the worst and any improvement, however small, is welcome!

Lostock Software are at 13, Cranborne Close, Lostock, Bolton, Lancs BL6 4JG

COMMAND	**FUNCTION**
Control-W	Call the Editor
Control-W,A,S,Z	Move the copy cursor (Apple II)
Escape-I,J,K,M	Move the copy cursor (Apple II+ and Apple IIe
Escape-→,←,↑,↓	Move the copy cursor (Apple IIe)
→	Copy from copy cursor position
←	Delete characters in input line
Control-E	Copy to end of screen line
Control-Q	Ignore LISTing format's extra spaces
Control-P	Revert to LISTing format
Control-L	Output LIST

AUTO LINE NUMBERER COMMANDS

Control-A	Intialise auto numbering: START: first line number (10) INC: increment (10)
Space bar	Output next line number
Control-O	Disable auto numbering (Note O not 0)

Dr. Barry Landsberg

SOME NOTES ON THE APPLE

Last month we looked at a superior method of generating music by machine code routines for the 6502. In the second part of the series we put a bit of life into the notes by providing improved tone.

Have all of you APPLE owners typed in the programs from last month's article and listened to the 'Pizzicato Polka' and the 'Toccata'? Perhaps you may have even programmed a vast range of other tunes already! If so, once you have got over the initial joy of extending the musical range of the Apple, you might have noticed that the tone quality of the music produced still sounds somewhat hollow and empty. Furthermore, if you chose to examine my suggestion that the longest pause possible was given by a value of the time parameter T of 255 and put, say, 500 instead, you might have found that half the time you get an extraneous 'click' generated and half the time you don't — even though all the registers of the 6502 have been set to the same value in each case! The purpose of this article is to provide an explanation of these points, and also an enhancement to the tone quality of the music generated by the APPLE using software alone.

The machine code programs published in last month's article all produce music by waiting for a certain specified period of time and then accessing the address $C030, which according to the Apple Reference Manual changes the state of the loudspeaker either from 0 V to 5 V, or from 5 V to 0 V, depending on its previous state. This process is repeated over and over again for the duration of the note and produces a square-wave with a mark-to-space ration of 1. A Fourier series (which is just a mathematician's way of splitting up a periodic waveform into a sum of sine waves) on this kind of square-wave shows that the nth harmonic has an amplitude of $(\sin(n\pi/2))/n$, from which it follows that only the odd harmonics are present, and the first, third, fifth . . . harmonics have amplitudes of 1, 1/3, 1/5 . . . respectively. This situation is shown diagrammatically in Fig. 1a, where the waveform is drawn followed by a graph showing the relative amounts of the first 15 harmonics. This analysis is made on the supposition that the loudspeaker generates the waveform that has been fed into it, which is not a valid assumption especially at higher frequencies. Still, it does give a rough idea of what is happening. Throughout this article, the magnitude of the amplitudes will be taken, and their sign (ie whether positive or negative) will be ignored. The human ear is relatively insensitive to phase anyway. It is the absence of even harmonics that is the hallmark of musical notes generated in such a fashion by the computer, and that gives rise to the characteristic hollowness of each note.

RAISING THE TONE

Having demonstrated that a square wave gives rise to a hollow-sounding note, but that the loudspeaker on the Apple has only two voltage levels, how can we possibly enrich the sound of each note without having to resort to a digital synthesis method? One way to do it is to still produce notes using a square wave, but with a mark-to-space ratio not equal to 1. Consider what would happen if a mark-to-space ratio of 1:3 were to be generated. The Fourier series analysis shows that the nth harmonic now has an amplitude equal to $(\sin(n\pi/4))/n$, and although there is no harmonic that is a multiple of 4 present, there are still plenty of even harmonics in the note, and it will certainly have a different tone quality to that of the normal square wave! Figure 1b shows this situation diagrammatically, and the result would be the same had the mark-to-space ratio been 3:1 instead.

Now that we have the hint that this kind of waveform might produce a more pleasant sound, how do we program it under the very stringent condition that it must be compatible with, and easy to integrate into, the music package given in last month's article? All kinds of general mark-to-space algorithms may be sought, but as these would involve extra counters so that the routine knows exactly when to strobe the loudspeaker and when not to, the timing of the loops controlling the pitch would be affected and the result would be less than musical! One way round the problem is to write a routine specifically for a 1:3 mark-to-space ratio by having four identical routines like that of Listing 2 in last month's article, and to jump from one to the next in a cyclic fashion. In addition, the last two routines must not strobe the speaker, but must take up exactly the same time as those that do. To do this, the instruction BIT $C030 (which takes four clock cycles and strobes the speaker) is replaced by two NOP instructions which do nothing at all, but also take four clock cycles.

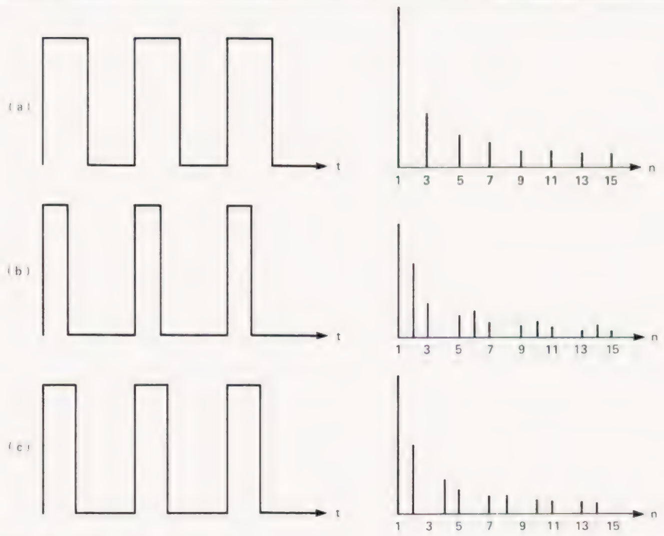


Fig. 1 Square waves with mark-to-space ratios of 1:1 (a), 1:3 (b) and 1:2 (c). Also plotted are the amplitudes of the first 15 harmonics of these waveforms as calculated by a Fourier series analysis.

The machine code routine to do this is given in Listing 6, and starts at address $0358, which is immediately following that of Listing 4 last month. It should be noted that Applesoft and DOS use an area of memory starting from $03D0 to store some of their jump vectors, and so the whole music package tucks into this available area of memory with only a few bytes to spare! The result of all this is that we now have a routine which plays an enriched sound, and which may be invoked by calling subroutine $0358 instead of $0300. However, doing this gives rise to a note that is an octave below that generated by the routine at $0300.

To integrate this program into the package presented last month we now only need to alter some CALL statements or do a quick POKE! For example, we merely replace the command CALL 768 in line 60 of Listing 3 with CALL 856 in order to play a harmonically enriched version of the 'Pizzicato'. However, due to the nature of Listing 5 it is played an octave lower, but by adding IF I THEN I=I+12 at the end of line 30 we can bring the Pizzicato back to the correct pitch. Try comparing the two and see for yourselves which is preferable!

The general music package described last month calls Listing 5 to set up the musical buffer, and then calls Listing 4 to step through the buffer and produce the melody. In order to make it generate the enriched sound described above, it is only necessary to alter the command JSR $0300 command in Listing 4 to JSR $0358. This may be done either from the Apple's monitor, or more easily by the command POKE 851,88 which must be executed before CALL 799 is reached.

It is unfortunate that the 'Toccata' given as a demonstration of the ability of Listing 5 is not the best example for this enriched sound routine, as the bottom note is already so low that making it yet an octave lower reduces it to well below that acceptable as a musical note. Neither may we transpose it up an octave as was done for the 'Pizzicato' because the 'Toccata' already spans three and a half of the available four octaves. Still, if the bottom notes may be tolerated, the rest of the 'Toccata' is certainly enriched. The lesson to be learnt from this is not to use anything below note 12 when calling Listing 6.

In summary, the command POKE 851,88 ensures that Listing 5 plays the melody in the musical buffer using the waveform with a mark-to-space ratio of 1:3, while POKE 851,0 gets the melody to be played using the normal square wave.

GETTING RICHER

The reader may wonder at this stage why I picked out a mark-to-space ratio of 1:3 instead of any other ratio. The reason is that, while the normal square wave needs two traversals of the timing loops for each cycle, the 1:3 ratio needs four. The periodicity (and thus the resultant frequency) is therefore halved, and the melody is played an octave lower. No change of key signature is involved, and the optional IF I THEN I=I+12 is only to alter the octave at which the melody is pitched. Any other simple ratio of small whole numbers results in a transposition of the music to another key, or sometimes even to musically non-existent tunings.

Having gone to all the trouble of setting up a program which generates a mark-to-space ratio of 1:3, is there any way to easily change the mark-to-space ratio? Is there in fact any need to? Well, after a while, even the 1:3 ratio itself starts to sound empty and the ear craves yet further variety. Perhaps the lack of fourth harmonic and the dominance of odd harmonics in the waveform has something to do with this! It turns out to be very easy to convert Listing 6 into a generator of square waves with a 1:2 mark-to-space ratio. Now the nth harmonic has an amplitude of $(\sin(n\pi/3))/n$: this is shown pictorially in Fig. 1c. However, this wave shape has a periodicity of 3 cycles instead of 2, and thus the notes produced have two-thirds of the frequency of the regular square wave. This turns out to be a musical interval of a fifth, and thus any pitch parameter P produces a note seven semitones lower than would be produced by Listing 2.

How then do we convert Listing 6 to produce a 1:2 mark-to-space ratio? We must bear in mind that Listing 6 consists of four similar routines where only the first two strobe the speaker, and all four are stepped through in a cyclic fashion. It can be fairly easily seen that stepping through the first three only gives rise to the desired waveform, and this is done by replacing the instruction JMP $03B3 (which causes a jump from the third routine to the fourth) with JMP $035A, which jumps back to the first one instead. This may be done under program control with the instruction POKE 943,90 which converts the 1:3 waveform to a 1:2 waveform, and the command POKE 943,179 may be used to regenerate the 1:3 ratio again.

If the commands POKE 851,88 : POKE 943,90 are typed into Listing 5, the 'Toccata' is now played in G minor (sorry, Bach!), and the above commands followed by IF I THEN I=I+7 typed in to line 30 of Listing 3 produce the 'Pizzicato' with the 1:2 waveform. It is my opinion that this gives rise to the sweetest sounding of the notes produced by the waveforms discussed in this article.

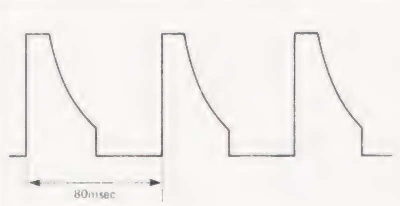


Fig. 2 The waveform of the voltage across the loudspeaker of an Apple as measured on an oscilloscope. The frequency of the waveform is about 13 Hz. Note that the flat portion lasting about 15 milliseconds is followed by a decay with a half-life of about 10 milliseconds.

In summary, a complete musical package has been given in which not only is the melody easily programmable as explained in last month's article, but by using only one or two POKE commands, the waveform may be chosen to have a mark-to-space ratio of 1:1, 1:2 or 1:3. This means that the tone quality may now be chosen for any melody, and even changed within the melody, somewhat analogous to pulling stops on an organ. This is surely a minimum that should be expected from any set of programs which purport to play music that is actually pleasant to listen to!

THE SPEAKER STROBE

The remainder of this article is concerned with a feature of the strobe to the Apple's loudspeaker, which caused me great confusion at first while I was writing and testing the machine code musical routines. The Apple Reference Manual confidently asserts that every time the address $C030 is accessed, an audible click is generated by the loudspeaker. Is this true? Type in the following program:

```
10 FOR I=1 TO 10
20 X=PEEK(-16336) : REM ** STROBE
   $C030
30 FOR J=1 TO 100 : NEXT J : REM
   ** DELAY
40 NEXT I
```

How many clicks do you hear when you run it? Surely we must expect 10, but I assure you that there are only five! However, if one makes frequency measurements on musical notes produced by the routines in this series and does simple timing calculations, it is evident that there actually is one strobe to the speaker for each and every access to the address $C030.

This seemingly contradictory state of affairs was brought to light while trying to understand two observations made about the musical routines. The first is that a pause with a time parameter between 256 and 511 sometimes produced an unwanted click and sometimes didn't, as mentioned earlier. The second is the more subjective observation that sometimes low notes sounded worse than at other times when using the waveform with a mark-to-space ratio of 1:3.

The explanation of all this is to be found in the circuit diagram on Page 114 of the Apple Reference Manual. The capacitor C11, along with the 47k resistor and diode, can alter the incoming square wave to a step followed by a voltage decay for a positive-going step, and by a drop to 0 V until the appearance of the next strobe for the negative-going step. This waveform is now the input to a Darlington transistor pair which actually becomes saturated immediately after a positive-going step.

A diagram of the voltage across the speaker as a function of time is shown in Fig. 2. This was determined from a (so-called) square wave of about 13 Hz, and was measured on an oscilloscope. It is evident that there is no voltage decay for the first 15 milliseconds following the strobe (corresponding to the period of time for which the Darlington pair is saturated), and thereafter the voltage decays to zero in an exponential fashion with a half life of about 10 milliseconds.

It is now clear why only five clicks were heard in the above program, and why some pauses gave extra clicks only half of the time — it is because when the loudspeaker is left in a state with 5 V across it, it decays to 0 V within 100 milliseconds. After this time the next strobe does not have any effect on the loudspeaker, even though the controlling flip-flop does change its state! Finally, low notes with an uneven mark-to-space ratio may sound different depending upon whether the longer of the two delay times between loudspeaker strobes corresponds to 0 V across the speaker or not.

This article extends the musical package presented last month to include a method of altering the basic waveform of the notes produced and thus a corresponding change in the tonal quality of the music generated by the Apple. It also warns of possible pitfalls that may be encountered at lower frequencies due to hardware considerations. I hope these articles will catalyse readers into doing further musical experiments on their Apples — I certainly shall be!

```
2  DATA 65279,768,724,686,646,610,576,544,514,485,458,432,408
3  DATA 384,362,342,322,304,287,271,256,241,228,215,203
4  DATA 191,180,170,160,151,143,134,126,119,113,106,100
5  DATA 95,89,84,80,75,71,67,63,60,56,53,50,47,44
10  GOSUB 1000
20 AD = 16384:TF = 256:UN = 1
30  READ I,T
40 P = N(I):P1 =  INT (P / TF) + UN:T1 =  INT (T / TF) + UN
50 P2 = P - TF * INT ((P + UN) / TF) + UN:T2 = T - TF * INT
   ((T + UN) / TF) + UN
60  POKE AD,P1: POKE AD + 1,P2. POKE AD + 2,T1: POKE AD + 3,T2:AD = AD + 4
70  IF T THEN 30
100  CALL 799. END
130  DATA 29.300,32,200,40,100 41,600
140  DATA 36,100,39,100,37,100,32,200,31,100,30,550,0,50
150  DATA 30,300,32,200,34,100,36,600
160  DATA 34,100,32,100,31,100,30,200,39,100,37 600
500  DATA  0,0
1000  DIM N(50)
1010  FOR I = 0 TO 50. READ N(I): NEXT I
1020  RETURN
```

Listing 6.

```
2000-   A9 00      LDA  #$00   DISENABLES STROBE TO SPEAKER
2002-   8D 1A 03   STA  $031A , IN SUBROUTINE $0300
2005-   A5 FE      LDA  $FE
2007-   85 EE      STA  $EE
2009-   A5 FF      LDA  $FF
200B-   85 EF      STA  $EF
200D-   A9 01      LDA  #$01
200F-   85 FE      STA  $FE
2011-   A9 16      LDA  #$16
2013-   85 FF      STA  $FF
2015-   20 58 03   JSR  $0358 , PLAY NOTE FOR #$16 UNITS OF TIME
2018-   38         SEC
2019-   A5 EF      LDA  $EF
201B-   E9 16      SBC  #$16
201D-   B0 04      BCS  $2023
201F-   C6 EE      DEC  $EE
2021-   F0 19      BEQ  $2040
2023-   85 EF      STA  $EF
2025-   A9 01      LDA  #$01
2027-   85 FE      STA  $FE
2029-   A9 0B      LDA  #$0B
202B-   85 FF      STA  $FF
202D-   20 00 03   JSR  $0300 , SILENCE FOR #$0B UNITS OF TIME .
2030-   38         SEC
2031-   A5 EF      LDA  $EF
2033-   E9 0B      SBC  #$0B
2035-   B0 04      BCS  $203B
2037-   C6 EE      DEC  $EE
2039-   F0 05      BEQ  $2040
203B-   85 EF      STA  $EF
203D-   4C 0D 20   JMP  $200D
2040-   A9 C0      LDA  #$C0  , RE-ENABLES STROBE TO SPEAKER
2042-   8D 1A 03   STA  $031A , IN SUBROUTINE $0300
2045-   60         RTS
```

Listing 7.

```
0358-   A9 00      LDA  #$00
035A-   A6 FD      LDX  $FD   . FIRST SECTION
035C-   A4 FC      LDY  $FC
035E-   69 01      ADC  #$01
0360-   D0 08      BNE  $036A
0362-   C6 FF      DEC  $FF
0364-   D0 04      BNE  $036A
0366-   C6 FE      DEC  $FE
0368-   F0 5E      BEQ  $03C3
036A-   CA         DEX
036B-   D0 F1      BNE  $035E
036D-   88         DEY
036E-   D0 EE      BNE  $035E
0370-   2C 30 C0   BIT  $C030 . STROBES LOUDSPEAKER
0373-   4C 76 03   JMP  $0376
0376-   A6 FD      LDX  $FD     SECOND SECTION
0378-   A4 FC      LDY  $FC
037A-   69 01      ADC  #$01
037C-   D0 08      BNE  $0386
037E-   C6 FF      DEC  $FF
0380-   D0 04      BNE  $0386
0382-   C6 FE      DEC  $FE
0384-   F0 42      BEQ  $03C3
0386-   CA         DEX
0387-   D0 F1      BNE  $037A
0389-   88         DEY
038A-   D0 EE      BNE  $037A
038C-   2C 30 C0   BIT  $C030 , STROBES LOUDSPEAKER
038F-   4C 92 03   JMP  $0392
0392-   A6 FD      LDX  $FD   . THIRD SECTION
0394-   A4 FC      LDY  $FC
0396-   69 01      ADC  #$01
0398-   D0 08      BNE  $03A2
039A-   C6 FF      DEC  $FF
039C-   D0 04      BNE  $03A2
039E-   C6 FE      DEC  $FE
03A0-   F0 26      BEQ  $03C8
03A2-   CA         DEX
03A3-   D0 F1      BNE  $0396
03A5-   88         DEY
03A6-   D0 EE      BNE  $0396
03A8-   EA         NOP         . DOES NOT STROBE LOUDSPEAKER
03A9-   EA         NOP
03AA-   4C AD 03   JMP  $03AD
03AD-   A6 FD      LDX  $FD     FOURTH SECTION
03AF-   A4 FC      LDY  $FC
03B1-   69 01      ADC  #$01
03B3-   D0 08      BNE  $03BD
03B5-   C6 FF      DEC  $FF
03B7-   D0 04      BNE  $03BD
03B9-   C6 FE      DEC  $FE
03BD-   F0 0B      BEQ  $03C8
03C3-   CA         DEX
03BE-   D0 F1      BNE  $03B1
03C0-   88         DEY
03C1-   D0 EE      BNE  $03B1
03C3-   EA         NOP         . DOES NOT STROBE SPEAKER
03C4-   EA         NOP
03C5-   4C 5A 03   JMP  $035A , JUMPS BACK TO FIRST SECTION
03C8-   60         RTS
```

Listing 8.


MOGUL
THE HOTWARE PEOPLE
MOGUL
METAMORPHOSIS
MOGUL
CREATORS REVENGE
METAMORPHOSIS
You stumbled into the nest of the Cyglorx and find yourself fighting off robot tanks guarding the Cyglorx eggs. You think you have everything under control and then the eggs start hatching. Commodore 64 version features 4 different screens.
VIC 20 – COMMODORE 64
£7.95
CREATOR'S REVENGE
The creator assembled a massive army of robots and insects to take revenge on the earth. Destroy insects, get treasures, and get the neutron bomb deactivator. Battle robots and destroy the neutron bomb before it annihilates your city. Miss and you must face the mutants. Features 4 different screens.
COMMODORE 64
£7.95
OTHER GAMES AVAILABLE
KONGO KONG
ANNIHILATOR
ANNIHILATOR 2
SUPER TREK
PYRAMID
ZEUS
POLLYWOG
GREAT ADVENTURE PACK
EARTHQUAKE
MOGUL COMMUNICATIONS LIMITED
90 Regent Street, London W1R 5PT Telephone 01-437 3156/7

Don Thomasson

ZX81-FORTH

If you're fed up with your ZX81 and its Sinclair BASIC, it's possible to perform a bit of surgery and turn it into a different beast entirely. We review the multi-tasking ZX81-FORTH ROM from David Husband.

For lay observers, one of the oddest things about computers is the way they can be transformed into different machines by changing one small component. For example, it is possible to take a standard ZX81, fit a new ROM, and produce a computer specialised to the FORTH language.

Is that good? Opinions may vary, but answers to the question need to be based on more than a superficial glance at the subject, so let us probe a little more deeply.

FORTH seems to be the language of the moment, with fresh versions springing up everywhere. Unfortunately, many implementations are rather idiosyncratic, and there will soon be as many dialects of FORTH as there are of BASIC. That is a pity, because it inhibits the publication of source code programs. A program written for one implementation would probably be difficult to convert so that it would run on another version. The process of conversion might largely be a matter of defining new words, but that may need specialised knowledge.

ZX81-FORTH is radically different from most of its contemporaries. For one thing, it provides for multi-tasking. At a very simple level, this means that it can run a foreground program normally, but slip away to do other tasks at regular intervals. It would be possible to run a time display program or monitor input data from an external source concurrently with the execution of a main program.

At a time when newcomers to FORTH are struggling to come to terms with single-task systems, multi-tasking may seem an unnecessary complication, but use of the facility is not mandatory. The system can be used quite simply in the single-task mode.

THE HARDWARE

It is easy enough to talk about fitting a new ROM, but the process can present problems. Some ZX81s provide a socket for their ROM chips, some do not, the ROM being firmly soldered to the printed circuit board. The socket and the board layout provide for 28 pin connections, though the normal component has only 24 pins. The extra pins cater for the fact that an EPROM requires more connections than a pre-programmed equivalent — one to select program mode, another to supply the voltage used for programming. To complicate the matter still further, various manufacturers have different ideas regarding the position of the pins.

To overcome these problems, the FORTH ROM is supplied with a socket, and — if necessary — the ROM pins are 'modified', as the manual puts it. It all sounds slightly dicey.

Fortunately, it is possible to obtain ready-converted systems, and that might be a wiser course. The guarantee would not survive the process of home conversion. The situation regarding the guarantee for converted machines needs to be checked.

SWITCH ON

The point about the guarantee came to mind because the review unit had a problem, one seen before in ordinary ZX81s. Due to some confusion in the sync-generating system, there were two overlapping displays on the screen. It was possible to carry out some experiments, but only with some difficulty.

One point that emerged early on was that the Rubout only backspaced, the original text being altered only when it was overwritten. Worse, beyond about mid-screen the Rubout key had no effect. Combined with the need to key in each letter separately, this made the deficiencies of the ZX81 keyboard painfully evident.

The second point may have related to the hardware problem. A great advantage of having the FORTH kernel in ROM is that it is not destroyed when a program runs amok. User extensions, on the other hand, are not protected, and it was noted that they sometimes vanished, perhaps due to imperfect contact with the essential RAM extension box. Incidentally, another make of extension was tried, but that destroyed the display completely, yet both extensions worked on another ZX81.

A key problem (literally!) was that the ZX81 keyboard does not normally generate certain characters essential to FORTH. There is no @, no !, no square brackets, and so on. A conversion table is necessary, and a keyboard overlay would be useful to remind you to press OR for !, STEP for @, and so on.

These points are made as facts, not necessarily as criticisms. The rationale of using the ZX81 as a basis for conversion is economically sound, and the consequences must be accepted. Nevertheless, there was a feeling that a conversion of a slightly more expensive machine might have been preferable.

THE SYSTEM

Apart from the multi-tasking feature, the system departs from the norm in a number of significant ways. Instead of using a common stack for all data, separate numeric and character stacks are provided. There is presumably a Return stack as well, but it was not defined explicitly.

Another difference was the use of a split-screen arrangement to reserve the upper part for editing purposes, while the lower half became the 'console' screen. In usual FORTH practice there is no need for this, because the whole system is dedicated to one kind of task at a time. If editing is going on, the whole screen is dedicated to that task. If a program is being compiled, the screen only reports errors. If a program is being run, the whole screen is at its service. This straightforward situation changes when multi-tasking is in use.

A perennial problem with FORTH in association with a screen of limited size is the way in which adequate amounts of source code can be handled effectively. ZX81-FORTH provides a word SCREEN, which allows screen areas of various sizes to be set up, each area having its own name reference. There is also provision for copying a screen from one store area to another, and with practice this would probably serve the essential needs. The STORE command will pass the contents of the editor screen to tape, and LOAD will restore it. This could become rather tedious, since a number of small recordings will be involved, rather than a single overall file.

Because of these system characteristics, it is difficult to compare ZX81-FORTH directly with other versions of FORTH. It is said to be derived from TREE-FORTH, so it may not be completely out on a limb on its own(!), but it is certainly unusual in a number of ways.

VOCABULARY

A comprehensive FORTH kernel in Z80 code, using the normal machine

operating system to help out, runs to around 9 kbytes of store. It was therefore to be expected that a version incorporating its own operating system would be a tight fit in 8 kbytes, even without the addition of multi-tasking. This is quoted as a reason for departing from fig-FORTH standards, but the departure need not have been quite so radical.

Comparing ZX81-FORTH with a good fig-FORTH implementation, no more than one third of the words provided by the latter are common to both versions, even if functions with the same effect but different names are included. The differences work both ways. ZX81-FORTH caters for 64-bit products of 32-bit numbers, where fig-FORTH is usually limited to a 32-bit number length. The CASE structure is implemented, but in a way analogous to ON GOSUB in BASIC. On the other hand, some useful words are absent, or at least not defined in the 71-page manual, which is produced in A4 size on a good printer.

Some of the differences stem from the differences in system concept, but all add up to the fact that communication with other forms of FORTH is made more difficult than necessary.

So let us sit back and consider what ZX-81 FORTH offers. It is certainly not without interest. A correspondent in Florida, who uses a couple of ZX81s (he calls them TS1000s, of course!) to run a local radio station might well find the multi-tasking useful, though he would probably have to write some machine code definitions to access the input/output functions. The apparent absence of such functions is a pity, because FORTH is well-suited to control processes acting on external equipment.

Taking a thoroughly down-to-earth approach, let us pose the key question: Why should you decide to buy ZX81-FORTH? The first part of the answer is you can begin to experiment with FORTH for just under £75. (£45 ZX81 'starter pack' and £28.75 for the ROM). If you decide that FORTH is not for you, BASIC remains available, via a reverse conversion.

That will only be valid, however, if you have no computer to start with. If you have a ZX81, you can convert it for less than £30. If you have a Spectrum, you can convert it for £15, the cost of a FORTH tape, and that applies equally to a number of other machines.

Leaving aside the economics, ZX81-FORTH has two attributes which make it unique: the kernel in ROM and the provision for multi-tasking. These, frankly, are the two principal reasons which justify its existence. If it could have been kept nearer to other FORTH standards, there might have been more reasons.

CONCLUSION

The concept behind ZX81-FORTH is good, but the actual implementation could be better. For those who wish to explore FORTH, there is a danger that it will lead them into a dead end, with problems of communication with other FORTH users. On the other hand, those who have a particular application in mind, and who have the knowledge needed to make maximum use of the system, may find that ZX81-FORTH is exactly what they need.

The ZX81-FORTH ROM is available at £28.75 inclusive from:
David Husband, 2 Gorleston Road, Branksome, Poole BH12 1NW. (Callers by appointment: telephone (0202) 764724, 6-7 pm, Mon-Sat).
Converted machines, price to be announced, from:
Densham Computers Ltd, 329 Ashley Road, Parkstone, Poole, BH14 0AP. (Telephone (0202) 737493).

DRAGON 32

Hardware and Software

Available Now — Dragon 32 Expansion Unit

High quality design allows simultaneous connection of disc dr ve and/or up to four cartridges.

- Gold-plated connectors used throughout. Fully programmable cartridge selection. Allows High Speed Data Transfers. Easy to use. Instructions included.
- Price **£48.00** + VAT (Please add £2.50 post & packing)

Coming Soon A/D, D/A board, Eprommer, Graphics Board, etc. all fully compatible Dragon 32.

For Dragon 32

- Master Chess — Software Magazine for Chess Enthusiasts. Study and enjoy the games of the masters, past and present. Introductory selection with detailed audio commentary cassette.
- Price **£8** + VAT (plus £1 post & packing)

Send large SAE for details of future issues of Masterchess, subscription rates and our full range of Dragon 32 hardware products.

Cheques/Postal Orders payable to: **Prelectronics Ltd.**
Send to: **Prelectronics Ltd., Albro Castle, St. Dogmales, Dyfed, Wales.**

ORIC AND SINCLAIR COMPUTERS

Oric 1 computer 48K £143 (£141) £151 Oric colour printer £134 (£123) £140 Sinclair Spectrum 48K £131 (£131) £143. Spectrum 16K £101 (£105) £117 32K memory upgrade kit for 16K Spectrum (issue 2 only) £31 (£28) £30 Fuller Master Unit for the spectrum including speech unit, sound synthesizer, amplifier and joystick port £56 (£56) £62. Fuller full sized FDS keyboard for the spectrum with proper space bar £52 (£52) £62. ZX printer with 5 free rolls paper £41. ZX printer alone £36 (£38) £50. 5 printer rolls £13 (£16) £21. ZX81 £37 (£37) £47. Special offer pack ZX81 computer + 16K ram pack + game tape £49 (£55) £65. ZX81 16K ram packs £31 (£28) £30

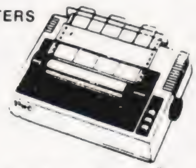
COMMODORE COMPUTERS

Commodore 64 £204 (£184) £204. Vic 20 £104 (£97) £117. Convertor to allow most ordinary mono cassette recorders to be used with the Vic 20 and Commodore 64:- built £9.78 (£9) £11. kit £7.47 (£7) £9 Commodore cassette recorder £43 (£44) £50. 1541 Disc drive £233 (£209) £234. 1525 Printer £235 (£220) £245. 1526 Printer £350 (£330) £360

ACORN COMPUTERS

Electron £203 (£209) £229 BBC Model B £424 (£388) £408. Kenda double density disk interface system for beeb £139 (£125) £135. We stock the whole range of Cumana disc drives for the beeb e.g. 100K single £230 (£220) £240. Double 2 x 400K £625 (£560) £580.

PRINTERS



Epson RX80 £326 (£309) £340. Epson RX80F/T £346 (£316) £346. Shinwa CTI CP80 £293 (£271) £312. FX80 £440 (£408) £438. Epson MX100/3 £494 (£465) £495. Seikosha GP100A £234 (£219) £254. Oki Microline 80 £223 (£207) £248. Oki Microline 84 £831. The Ultra 21 combined daisy wheel and electric typewriter £438 (£415) £445. The brother EP22 combined matrix printer and electric typewriter £173 (£166) £186. Juki 6100 proportional daisy wheel printer £423 (£404) £434. MCP40 colour printer £134 (£123) £140. Star STX80 thermal printer £165 (£159) £169. We can supply interfaces to run all the above from Sharp computers £58 (£52) £55

VIDEO GENIE, UK101 and SUPERBOARD

We still support these gorgeous machines. Write for our tempting list of add ons

SWANLEY ELECTRONICS

Dept CT, 32 Goldsel Road, Swanley, Kent BR18 8EZ, England.
TEL: Swanley (0322) 64851

Nothing extra to pay. All prices are inclusive. UK prices are shown first and include post and VAT. The second price in brackets is for export customers in Europe and includes insured air mail postage. The third price is for export customers outside Europe (include Australia etc) and includes insured airmail postage. Official orders welcome.

COMPUTER ROBOTICS TRAINING

FULL TIME COLLEGE COURSE

SUITABLE FOR APPLICANTS WHO WISH TO ENTER COMPUTER SERVICE OR RELATED INDUSTRIES — HIGH PERCENTAGE OF PRACTICAL COURSE WORK

15 MONTHS
B TEC Certificate in Computing Technology

9 MONTHS
B TEC Higher Certificate in Computing Technology

Subjects: Foundation Electronics, Digital Techniques, Microelectronics, Microprocessors, Microcomputer Based Systems, Industrial Robotics, Machine Code & High Level Programming.

Shortened courses can be arranged for applicants with previous knowledge.

Courses commence Jan, April and Sept (Higher Cert Sept only). Prospectus from:

LONDON ELECTRONICS COLLEGE (Dept C5/6)
20 Penywern Road,
Earls Court, London SW5 9SU.
Tel: 01-373 8721

TEXAS TI994A

NEW

Adventure No. 1
Hangman
Address Book
Home Accounts
Optical Illusions

all at £4.50 each

including P&P

Buy any 2 and receive Optical Illusions **FREE**

LINCO DATA

30 BROWN CRESCENT,
SUTTON-IN-ASHFIELD,
NOTTINGHAM NG14 7GY.

M. C. Hart

TWO PET UTILITIES

We proudly present a pair of PET utilities. There's a PRINT USING facility for the formatting of numerical data, and a program protection routine that should fool most of the people most of the time.

Now that PET computers in one version or another have been with us for some five years and a fair volume of material has been published about them, I speculated upon the type of articles that programmers of CBM machines most like to read. Having given the matter some thought, I suspect that many readers will fall into either or both of the following categories:

- those who enjoy elegant programming that makes use of CBM specialties, for example ROM routines.
- those who would like a 'utility' to complement the existing ROM routines, which are now starting to look a little threadbare when compared with what is available on more up-to-date machines such as the BBC.

Two routines are presented here, each of which has attempted to be brief yet practical. The first routine is a PRINT USING facility which was an amazing omission on machines intended for commercial use, while the second is a fairly simple way to give program protection to cassette based software.

PRINT USING UTILITY

While 'quick and dirty' methods of rounding to a specified number of digits exist, I was anxious to write a routine which was economical, fast and performed the following functions:

- to round both positive and negative numbers correctly, avoiding the rogue errors that the PET evaluator will introduce when processing certain numbers by conventional means (such as trying to round 812.676144 by using the INT(X*1000+0.5)/1000 approach).
- to process numbers less than ±0.01 which would otherwise be expressed in exponential format.
- to put in leading zeroes for values between +1 and −1.
- to add a fractional part of trailing zeroes to ensure consistency with other output: for example, 2.3 becomes 2.300 to three decimal places.

The routine presented here does all of these things and has attempted to optimise on a combination of speed and compactness. Timed on a BASIC IV (12"screen) machine, each subroutine call takes some 0.075 seconds, some 13 numbers to the second, and I believe that it compares very favourably with the machine code routine in the COMMAND-O chip which takes on average 0.047 second. (The latter, incidentally, will not attempt to process numbers in exponential format fully and for certain numbers is actually inaccurate, so 0.001 to three decimal places is processed as −03.000! Commercial or scientific users of the PRINT USING routine on the COMMAND-O chip be warned — you may be better off with my routine.)

Here are a few words of explanation for those who are interested in the construction process. Line 10 initialises variables as near to the start of the program as possible. As each of these variables is going to be used constantly in the formatting subroutine, access time needs to be fast to aid speed of execution. This is ensured by defining them in the variable list before other program variables are used. The interpreter therefore does not have to search far to find the subroutine variable names. Notice also that variable names rather than raw numbers are utilised — this means that some of the conversion time spent in converting numbers to the format in which they are stored internally is saved.

Line 11 constructs a small look-up table (not dimensioned unless you intend more than 10 places of decimals!) which contains the powers of 10 at each point (ie. 1,10,100 etc). The reason for this is that exponentia-

ORIGINAL VALUE	FORMATTED VALUE	
44.7437833	44.7438	44.74
1.45608437	1.4561	1.46
1256.02015	1256.0201	1256.02
-2927.99181	-2927.9918	-2927.99
417.395913	417.3959	417.40
4.98288499	4.9829	4.98
5.00350826E-03	0.0050	0.01
-.273648306	-0.2736	-0.27
7.5962991	7.5963	7.60
14.0696188	14.0696	14.07
760.401918	760.4019	760.40
-.478110656	-0.4781	-0.48
.0940358311	0.0940	0.09
.0316467338	0.0316	0.03
.0416842159	0.0417	0.04

Listing 2. A sample printout using PRINT USING (!).

```
1 PRINTCHR$(147)TAB(12)
2 PRINT"PRINT USING DEMO":PRINTTAB(13)
3 PRINT"================":PRINT:PRINTTAB(15)
4 :
5 PRINT"BY M.C.HART":PRINTTAB(15)
6 PRINT"===========":PRINT:PRINT:FORZR=1 TO 1500:NEXT
7 PRINTCHR$(147)CHR$(18)"ORIGINAL VALUE        FORMATTED VALUE"
8 PRINT:PRINT
9 :
10 ZR=0:Z1=0:Z$="":ZH=.5:ZD=0:Z=0:ZF=11:ZZ$="000000":ZF$="          "
11 Z(0)=1:FORJ=1TO10:Z(J)=Z(J-1)*10:NEXT:REM TABLE OF ROUNDING NUMBERS
12 :
13 :
20 REM Z  = ORIGINAL NUMBER
21 REM Z$ = OUTPUT STRING
22 REM ZR = ROUNDING FACTOR
23 REM ZH = HALF-ADJUST(0.5)
24 REM ZD = NO OF DECIMAL PLACES
25 REM ZF = FIELD LENGTH
26 REM ZZ$= ZEROES FOR 'PADDING'
27 REM ZF$= BLANKS FOR 'PADDING'
28 REM Z()= TABLE OF ROUNDING NUMBERS
29 :
30 :
100 FORJ=1TO15:Z=EXP(RND(1)*14-6):IFINT(J/4)-J/4=0THENZ=-Z
110 PRINTZ;SPC(15-LEN(STR$(Z)));
120 ZD=4:GOSUB60000:REM NO CARRIAGE RETURN
130 ZD=2:GOSUB60000:PRINT:REM NEW LINE
140 NEXT J:END
150 :
160 :
60000 ZR=Z(ZD):Z1=INT(ABS(Z)*ZR+ZH)/ZR:IFZD=0THENZ$=STR$(Z1):GOTO60020
60010 Z$=STR$(INT(Z1))+"."+RIGHT$(ZZ$+MID$(STR$(INT((Z1-INT(Z1))*ZR+ZH)),2),ZD)
60020 PRINTRIGHT$(ZF$+LEFT$(STR$(Z<0),1)+MID$(Z$,2),ZF);:RETURN
60030 :
60040 :
```

Listing 1. The PRINT USING program.

tion is expensive of processor time the look-up table saves some 40% of the processing time when compared with the traditional method of exponentiation in arriving at rounding numbers. Because of the way in which numbers are stored internally, exponentiation may introduce other small errors as well.

Lines 20-28 document the function of the variables in the routine — all of these are preceded by Z so that the user can reserve these exclusively for the use of the subroutine to avoid the risk of corruption. Lines 100-140 are a small 'driver' routine to generate random numbers and process them. Notice that all one has to do is to specify the number of decimal places and then call the subroutine but if used in a conventional program one would have to copy the number to be processed into the variable Z before the call. The number to be processed is printed in the last line of the subroutine (again to save time) and has a delimiter to keep output on the same line for further processing. If you intend your **next** call to the subroutine to be on a new line then in the driver program remember to force a new line after the last subroutine call.

If speed is the absolute essence then it **is** possible to speed up the program even further. If you know that you are not going to process any integers then you can cut out the condition at line 60000 altogether (IF ZD = 0 . . .) and this might save some precious microseconds. To process negative numbers in a **slightly** faster way then one can introduce a new line as follows:

```
60015 IFZ<0 THEN Z$ = "-" +
   MID$(Z$,2)
```

and cut out the complex (ingenious?) way of processing the negative sign in the last line. This is the middle term of the concatenated string, ie LEFT$(STR$(Z<0),1): make this latter term merely Z$, so that the last line reads:

```
60020 PRINT RIGHT$(ZF$+Z$,ZF);:
   RETURN
```

Attempts to go much faster in BASIC may well be thwarted by the speed of one's printer rather than anything else and the virtues of the three-line version are that it is quite easy to type in if APPEND facilitates are not readily to hand.

Finally, you might like to have an indication of the output string being too long for the specified field length. This is actually very easy to program in a new line:

```
60015 IF LEN(Z$)<ZF THEN PRINT
   LEFT$("***************",ZF);:
   RETURN
```

and in this case your number would not be printed but the whole field would be filled by a symbol. Yet another approach is to 'tag' a symbol such as a % sign to the offending string and print that out instead.

PROGRAM PROTECTION

No program can ever be completely protected but I offer below a machine code subroutine which offers a **measure** of protection to cassette-based software. Its deterent effect is probably not very great for those whose knowledge of the internal routines of the PET is of a moderate standard, but I am sure that it will serve its purpose by providing a fair degree of protection.

In any protection program one wishes that the program will run normally, but that it cannot be LISTed and particularly that it cannot be SAVEd. This is achieved in the program offered here in the following way: when the program is RUN the first statement is a SYS call which alters the pointers to the start of BASIC, disables the stop-key (but in a way which does not disable the clock so that is still accessible for timing purposes), alters the CHRGET routines so that direct commands (except RUN) are 'rewarded' by resetting the machine, and finally sets a flag to show that the program is correctly entered. A further check routine is written which can be accessed periodically to ensure that the program is entered by the conventional route and any would-be 'burglar' will be rewarded by a machine reset which clears the contents of most of the RAM, including the main program.

Assuming the program you wish to protect is already resident in memory, try to ensure that the program cannot be 'crashed' in any evident way, eg by preventing division by zero or null INPUTs. Then make the first line:

```
1SYS1048"*** . . exactly 70
   asterisks
```

leaving no spaces between 1 and SYS or SYS and 1048 and then typing a **single** set of quote marks. The space provided by the asterisks is going to be filled by machine code eventually. Then periodically throughout the program use the following program line (it need only be typed once and then duplicated when necessary by giving new line numbers).

```
XX SYS655:REM"" (carriage return)
```

Now place the cursor over the second of the quote marks and use the INSERT cursor to open up a gap of some 15-16 spaces or more then fill these spaces up to the last quote mark with DELETE signs (reverse T in appearance) and cursor right over the final quote mark when you come to it. Finally type shifted left square bracket for BASIC IV (⊞ symbol), or shifted L for BASIC II (symbol) — these symbols prevent listing while the preceding delete signs make the line practically invisible (except for the final quote mark which remains). This operation sounds tricky but is actually very easy once you have performed it a few times. Having got one line correctly entered then duplicate a few more so that they are scattered throughout the program.

To enter the machine code, follow this procedure. Type SYS4 to break to the monitor and then M 040B 0451 checking that byte 0451 is a 0. If not, then you have probably made a mistake with the asterisks, so exit from the monitor and adjust until you get it right. Then type in the nine lines of machine code (70 bytes) from Listing 3a, not forgetting to press Return at the end of each line. Similarly, while still in the monitor enter the lines of

(a)

```
.:040B 14 14 14 14 14 14 14 14
.:0413 14 14 00 00 00 AD 8E 02
.:041B 85 A2 C9 A0 D0 2D A9 52
.:0423 85 28 20 D2 B5 78 A9 44
.:042B 85 90 A9 04 85 91 58 20
.:0433 8F 02 A9 4C 85 79 A9 9F
.:043B 85 7A A9 02 85 7B 4C E9
.:0443 B5 20 EA FF A9 FF 85 9B
.:044B 4C 58 E4 6C FC FF 00, 79
```

(b)

```
.:028F A5 91 C9 04 F0 03 6C FC
.:0297 FF A5 A2 C9 A0 D0 F7 60
.:029F 48 C9 9B F0 F1 68 4C A2
.:02A7 D3 20 20 20 20 20 20 20
```

(c)

```
.:040B 14 14 14 14 14 14 14 14
.:0413 14 14 00 00 00 AD 8E 02
.:041B 85 A2 C9 A0 D0 2D A9 52
.:0423 85 28 20 5B C5 78 A9 44
.:042B 85 90 A9 04 85 91 58 20
.:0433 8F 02 A9 4C 85 79 A9 9F
.:043B 85 7A A9 02 85 7B 4C 72
.:0443 C5 20 EA FF A9 FF 85 9B
.:044B 4C 31 E6 6C FC FF 00 79
```

(d)

```
.:028F A5 91 C9 04 F0 03 6C FC
.:0297 FF A5 A2 C9 A0 D0 F7 60
.:029F 48 C9 9B F0 F1 68 4C 02
.:02A7 E1 20 20 20 20 20 20 20
```

Listing 3. (a) First machine code data for BASIC IV machines. (b) Second machine code data for BASIC IV. (c) and (d) are the corresponding data for BASIC II machines.

code in Listing 3b by typing M 028F 02A7, thus putting this piece of code into the part of RAM which is actually part of the first cassette buffer.

Having put the machine code into the program that we wish to protect, we are now ready to SAVE it — but in a rather novel way and with one or two 'twists' to deter predators! We obviously wish to preserve the code that we have put into the first cassette buffer. However, if you have any knowledge of the PET operating system, you may know that every address in the first cassette buffer from 028F to 0339 has a $20 written into it after a 'normal' SAVE. We are going to exploit this knowledge because there **is** a way of saving the first cassette buffer, but most people do not know it and when they attempt to SAVE a program they will overwrite the machine code routine without which the program will crash!

This is how we SAVE the first cassette buffer routine. First of all, make up a name for your program and allocate it to a variable such as A$. It is very important that your name is 16 characters long and **that your final character is a shifted space.** You can check that your character is correct by typing PRINT ASC(MID$(A$,16)) and you should receive the answer 160. If you do not then retrace your steps at this point.

To save the machine code, we pop it into another variable such as B$ with the following direct mode command:

```
B$="": FOR J=0 TO 24: B$=B$ +
  CHR$(PEEK(655+J)): NEXT
```

Finally concatenate the two with A$ = A$ + B$ and then SAVE A$ (that's all!). You will see some weird effects on your VDU as machine code is interpreted as graphics symbols and some of the name may appear to be corrupted — but fear not, it will not reappear when reloaded and to all intents and purposes will appear as a 'normal' program.

Now when the program is loaded back again the name appears as normal but a LIST attempt will fail (actually the first line will be listed but it will be instantly deleted and you should be able to observe a momentary flash but not sufficient for you to read). A RUN command will now run the program quite satisfactorily and you can repeat this with a similar RUN, but the machine code ensures that LIST in direct mode will reset the machine once the program has successfully RUN. If you want to save a back-up copy of the protected program, you must do it after the program is LOADed but before it has been RUN. For back-up purposes in which you preserve the same program name then all you need to do is the following:

```
A$="": FOR J=0 TO 40: A$=A$ +
  CHR$(PEEK(639+J)): NEXT
```

followed, of course, by SAVE A$.

By now, you may be able to appreciate how the several parts of the machine code works. Once the program has been RUN then a LIST command will reset the machine. Before the program has been RUN then the program cannot be listed because the first line is made invisible and the three zero bytes are inserted to 'fool' the PET into thinking that the program is at an end. If an intrepid hacker breaks into the program then the 'invisible' SYS calls will stop the listing with a SYNTAX ERROR and the SYS calls themselves will 'reset' the machine if they detect that the program name does not contain the invisible shifted space character. A call into the first machine buffer with no machine code in it will crash the machine, so preventing a successful RUN. Even so, these protection devices which might appear to be complex only rely upon a certain degree of ignorance and no doubt can be circumvented by those in the know. It might be an interesting exercise to 'protect' a program and then offer it to friends to see if they are able to 'crack' it.

PRINTOUT

Dear Sir,

I was attracted to the August edition of Computing Today for its article 'Problem Page' for the Spectrum, but I am quite disappointed regarding the section on Prime Numbers.

In wht I believe is correctly called the 'the sieve of Eratosthenes' the method is to cast out multiples of primes a the theory states that all non-primes are multiples of primes; it is also described as the 'uniqueness of factorisation theory'.

Although Don Thomasson refers to this method he does not use it in his program, falling back upon the INTeger test for prims; an adequate but slow process.

I enclose a listing that correctly uses the (three times) faster method. It also includes '2' in the result as this is a prime also.

Lines 90 and 100 set up an array to extract all the prime numbers below the value 'x'. Line 110 sets loop 'i' for 2 TO SQR x. Line 120 converts the array to '1' for all multiples of 'i' including those controlling the first loop.

The result is that the first run through converts 4,6,8,10, 12,14,16,18, etc; the next additionally casts out 9,15,21,27, etc; then 25,75 etc; followed by 49,77 etc as 2,3,5,7 are successively used as values for 'i'.

It only remains to examine the array and print out the elements that are still set to 'Ø

I trust you find this helpful.
Yours faithfully,
John A. Mason, Frome.

```
10 REM ****************************
20 REM *      PRIME NUMBERS       *
30 REM *  Sieve of Eratosthenes   *
40 REM *        J A Mason         *
50 REM ****************************
100 INPUT "Number below which prim
es to be extracted ";x
110 DIM n(x): LET p=0
120 FOR i=2 TO SQR x: IF n(i) <> 0
THEN GO TO 140
130 FOR j=i*i TO x STEP i: LET n(j
)=1: NEXT j
140 NEXT i
150 FOR i=2 TO x: IF n(i)=0 THEN P
RINT TAB p;i;: LET p=p+8
160 NEXT i
```

Dear Sir,

The program for the Dragon 32 in your September issue to generate characters is an excellent program. I was able to type it in my TRS-80 Color Computer and get it to work straight away. I have been using it since to generate characters for use with other programs.

I did find one small error in the program. Line 435 which is used to generate the @ character in the graphics mode is incorrect (at least I couldn't get it to work properly). I corrected the line as follows:

435 R$(64) = "BM+6,0;L4H2U2E2 R2F2D1G1L2H1E1R1D2"

This information may be of use to your other readers.
Yours faithfully,
Charles S. Nichols,
Teddington.

Dear Sir,

Could you please bring to the attention of your readers that as a result of the increasing number of followers of that super adventure program on the Oric-1 called 'Hells Temple' from Kenema Associates, that we are forming a fan club. The object of the club is to spread our gospel amongst other Hells Temple players, to meet, correspond and generally mix it with others who have dared to enter the Temple and got the bug!

I'm sure we would all appreciate your kind assistance.

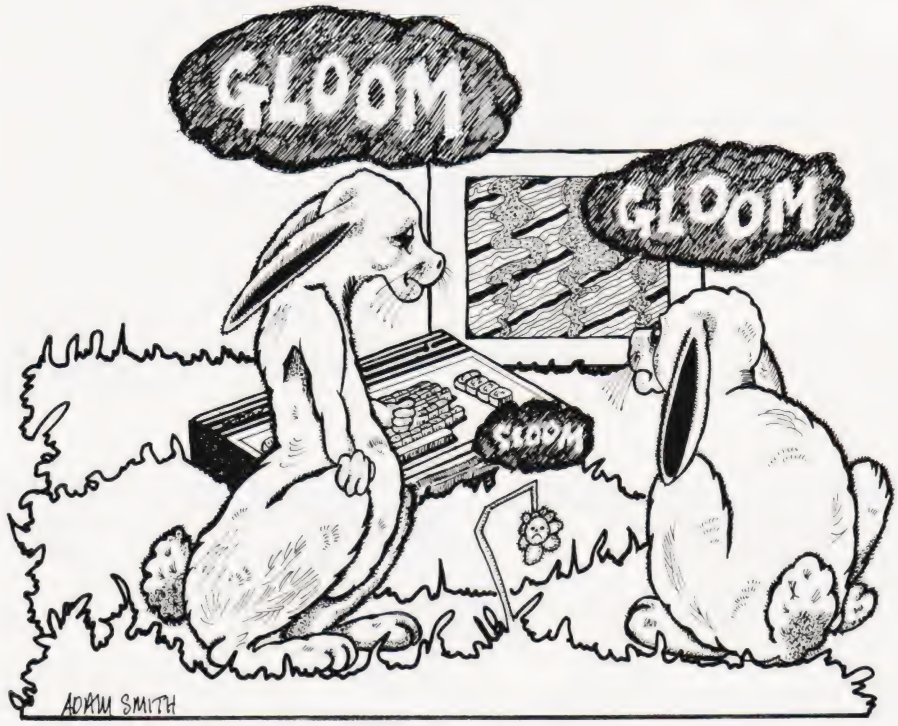
By the way, our club has been officially approved by Kenema Associates Limited.
Yours faithfully,
Dopple-Ganger.

(* Hmmm. . . Well, I suppose it takes all sorts. . . *)

Tony Cross and Phil Cornes

PROGRAM RECOVERY ON THE COMMODORE 64

Are you sitting comfortably? Then we'll begin. Like all good bedtime stories, this has a nasty beginning and a happy ending.



Once upon a time there were two programmers called TC and PC. One day they were happily programming their Commodore 64. When they had finished typing in a rather long program PC, who was always been a bit hasty, said "Go on, TC, type RUN and let's see if it works!" So TC typed RUN. After a few seconds the screen suddenly went blank and the keyboard stopped working. "Oh no!", cried TC, "It's hung on us! The only way out is to turn the power off". PC looked worried. "If we do that", he said, "we will lose the program we have just typed in!". "Oh dear", they moaned, "Oh dear" . . .

Go on — be honest — how many times has this happened to you? It's surprisingly easy to do on the Commodore 64 because so many of the sound and graphics facilities have to be POKEd in. On wrong number in a POKE statement could lead to just this sort of problem. One some machines you can recover from this problem by pressing a reset key which resets the CPU and then jumps into the monitor or warmstarts the BASIC. The Commodore 64 doesn't have a reset key — the only way to regain control of it once it has 'hung' is to turn the power off! (Pressing RUN/STOP and RESTORE doesn't always work).

To overcome this problem we decided to design a more elegant way of regaining control of a 'hung' Commodore 64 without losing the stored program.

A REAL RESET

Examination of the MOS 6510 chip reveals that it can be hardware reset by bringing the RESET pin low (earth). (The RESET is pin 40 of the chip). When the RESET pin goes high again (+5V) the 6510 loads its program counter with the contents of memory locations $FFFC and $FFFD. In other words, earthing the RESET pin will stop the CPU from doing whatever it is currently doing and will cause it to jump to the address contained in memory locations $FFFC and $FFFD.

The obvious thing to do is to load memory locations $FFFC and $FFFD with a vector which points to BASIC's warm start. Earthing the RESET pin will then stop the CPU and warmstart BASIC. Unfortunately, the KERNAL ROM overlays these locations, making it impossible to change them. The jump vector which the KERNAL maintains in $FFFC and $FFFD points to the normal 'cold start' routine. This is the routine used on power-on which resets all the BASIC pointers and does a RAM test before coldstarting BASIC. Earthing the RESET pin, then, regains control of the machine but 'loses' the current program.

BRING BACK BASIC

Actually things aren't as bad as they seem, because the program is still stored in memory, it's just that BASIC can no longer see it. What has happened is that the pointers, which tell BASIC where the program starts and ends, have been reset to indicate that there is no program in memory. (Fortunately the RAM test which has been performed is non-destructive which means that it hasn't damaged the program!). If we can find a way of restoring BASIC's pointers we can recover the program as well.

To understand how we can recover the program we need to look at the way BASIC programs are stored. On the Commodore 64 BASIC programs usually start at $0801 and can extend up to $9FFF. (Both these addresses can be moved but they rarely are). In addition, any variables and arrays are stored after the end of the program extending upwards, and strings are stored from $9FFF extending downwards. Naturally there are no variables, arrays or strings created until a program is RUN. Figure 1 shows the general storage of BASIC programs.

BASIC maintains pointers to several places, the four which interest us here are:

- The start of BASIC text. This pointer is called TXTTAB and it is at address $2B and $2C.
- The start of variable space. This pointer is called VARTAB and it is at address $2D and $2E.
- The start of array space. This pointer is called ARYTAB and it is at address $2F and $30.
- The end of array space. This pointer is called STREND and it is at address $31 and $32.

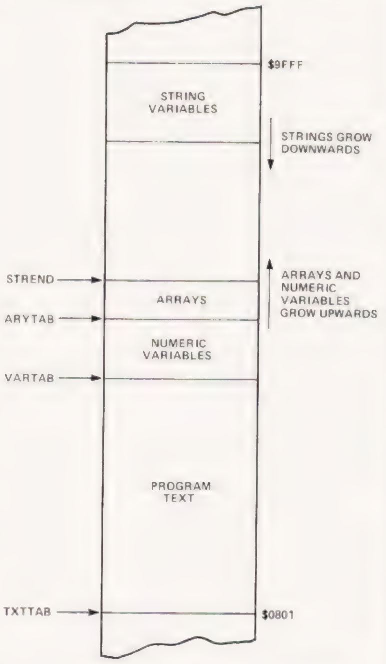


Fig. 1 The general storage method for BASIC programs on the Commodore 64.

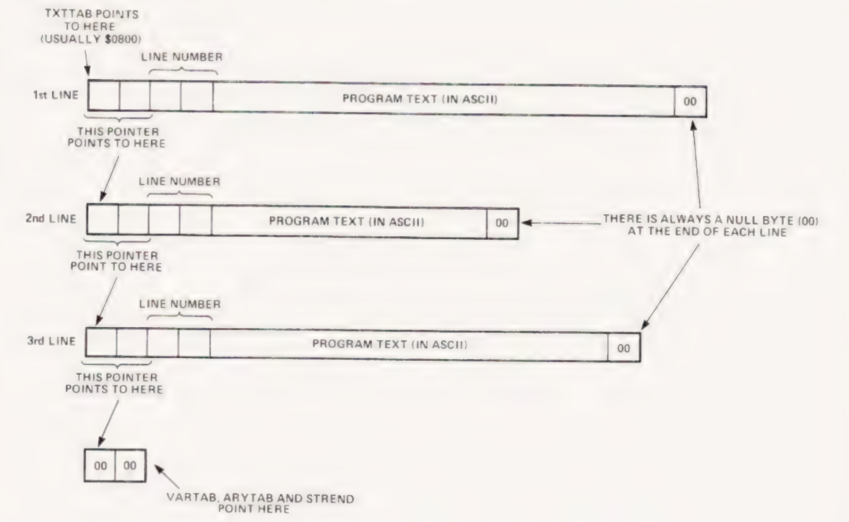


Fig. 2 How BASIC lines are stored in a three-line program.

Figure 1 also shows where these pointers refer to. Unless BASIC has been moved, TXTTAB will contain $0801, which is the normal start address of the BASIC text area. For a program which has never been RUN, VARTAB, ARYTAB and STREND will all point to the same place — ie the end of the program text.

When the power-on routine is executed, TXTTAB is set to $0801 and VARTAB, ARYTAB and STREND are all set to $0803 — ie no program in memory. What we have to do, then, is to find the end of the program and set VARTAB, ARYTAB and STREND to point to it.

Finding the end of the program is fairly easy because BASIC is stored as a linked list. This means that each line of a program contains a pointer to the next line. The last line in a program has two nulls ($00) in the pointer locations. Figure 2 shows how BASIC lines are stored in a short three-line program.

When the power-on routine is executed the first two pointer bytes are set to nulls — ie no program stored.

To find the end of the program, we first have to find the address of the start of the second line (by scanning for the null at the end of the first line) and set $0801 and $0802 to point to it. ($0801 and $0802 are the line pointer locations for the first line). We can then simply run down the pointers looking for the two nulls at the end of the program. The address of the byte immediately following them is the address which must be poked into VARTAB, ARYTAB and STREND.

```
10 FOR X=0 TO 84
20 READ Q : POKE 49152+X,Q
30 NEXT X
40 END
50 DATA 169,1,133,253,169,8,133,254
60 DATA 160,4,169,0,209,253,240,4
70 DATA 200,56,176,248,200,152,160,0
80 DATA 24,101,253,145,253,200,169,0
90 DATA 101,254,145,253,160,0,177,253
100 DATA 208,27,200,177,253,208,22,169
110 DATA 2,24,101,253,133,45,133,47
120 DATA 133,49,169,0,101,254,133,46
130 DATA 133,48,133,50,96,160,0,177
140 DATA 253,170,200,177,253,133,254,134
150 DATA 253,56,176,208
```

Listing 1. The BASIC program to POKE the machine code into memory. It is completely relocatable.

MAKING THE MODS

We now have enough information to enable us to implement the reset facility. Our first task is to wire up the reset key itself. The key can be of any type you like, although a subminiature 'push-to-make' type is probably the most suitable. It must be connected to the Commodore 64 so that when operated it connects an earth to the RESET track.

There are two ways of doing this: the first is to actually mount the

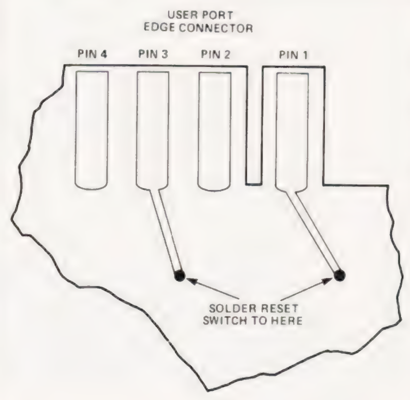


Fig. 3 Solder point locations for fitting a reset switch.

Listing 2. The assembly listing of the OLD program.

```
;OLD routine for the Commodore 64
;
;
;Copyright (c) A.L.Cross 1983
;
;
        *=      $C000       ;Set routine start
;
;
;Variables and equates
;
TEXT    =       $0801       ;BASIC prog start
VARTAB  =       $2D         ;BASIC variable  start
ARYTAB  =       $2F         ;BASIC arrays start
STREND  =       $31         ;BASIC strings end
LINPTR  =       $FD         ;Pointer
;
;
;First initialise LINPTR to point to the start
;of the BASIC program.
;
OLD     LDA     #<TEXT      ;
        STA     LINPTR      ;Copy prog start
        LDA     #>TEXT      ;address to LINPTR.
        STA     LINPTR+1
;Now restore the pointer on the first line of
;the program. (It must point to the second line).
;
        LDY     #$04        ;Initialise index.
        LDA     #$00        ;Search character.
INITLP  CMP     (LINPTR),Y ;
        BEQ     ENDEND      ;Scan for end of
        INY                 ;first BASIC line.
        SEC                 ;(Look for null).
        BCS     INITLP      ;
ENDEND  INY                 ;Step over null.
        TYA
        LDY     #$00        ;Initialise index.
        CLC                 ;Calculate pointer
        ADC     LINPTR      ;value.
        STA     (LINPTR),Y ;Put low byte back.
        INY
        LDA     #$00
```

switch in the case at some convenient point and then wire it in to earth and RESET. The most convenient place to pick up these signals is on the User port. If you take the top off the computer you will see that there are two convenient soldering points just behind pins 1 and 3 of the User port. Pin 1 is the earth and pin 3 is the RESET lead. Figure 3 shows the location of these solder points. Be warned, however, that this will almost certainly invalidate any guarantee on the machine.

The second method of connecting the reset key, which may not invalidate the guarantee is to wire the switch to pins 2 and 6 of a DIN plug. The plug and switch can then simply be plugged into the Serial port. (The earth and RESET leads appear on pins 2 and 6 of the Serial port). Figure 4 shows a simple diagram of these two methods.

Now on to the software. This must, of course, have been loaded before the reset key is used (it's too late after you've pressed it!). We have presented the routine, which we have called OLD, in two forms. Listing 1 is a BASIC program which will POKE the routine in, and Listing 2 is a 6510 assembly listing of the same routine with comments so that you can see how it works.

The routine is only 84 bytes long and it can be relocated anywhere you like. (Locations below $0300 are overwritten during the power on routine). The routine is called by a SYS 49152 command in our version and it only takes a few milliseconds to run.

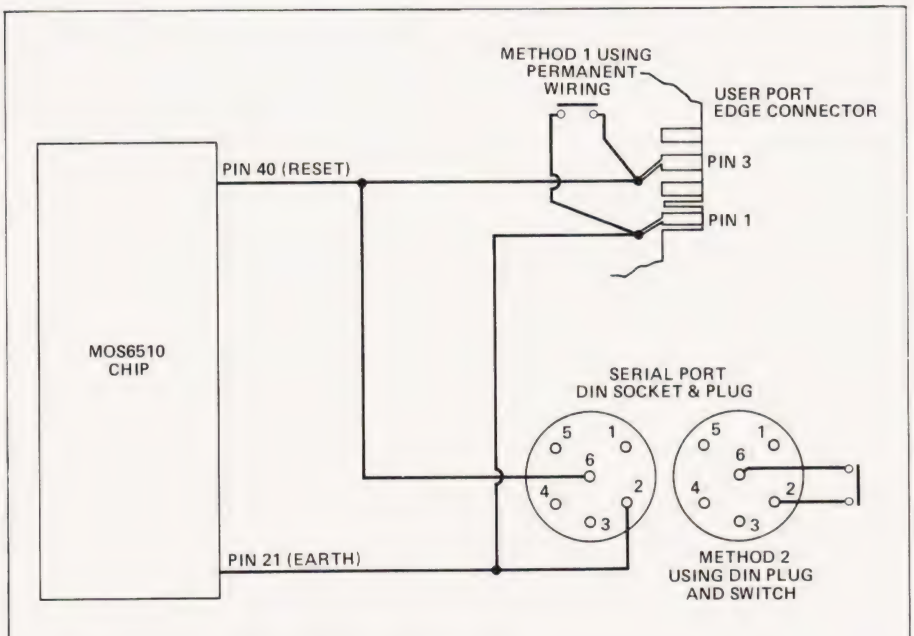


Fig. 4 The two methods of wiring the switch

CONCLUSION

The Commodore 64 now has a very useful reset key which will always reset the processor from any state. In addition the OLD routine will recover the current program, provided that the bug which caused the processor to 'hang' has not overwritten any important memory locations.

Incidentally, the normal BASIC NEW command works by simply resetting BASIC's pointers, so if you accidentally type NEW, a call to our OLD routine will restore the program for you! (Now you know why the routine is called OLD!).

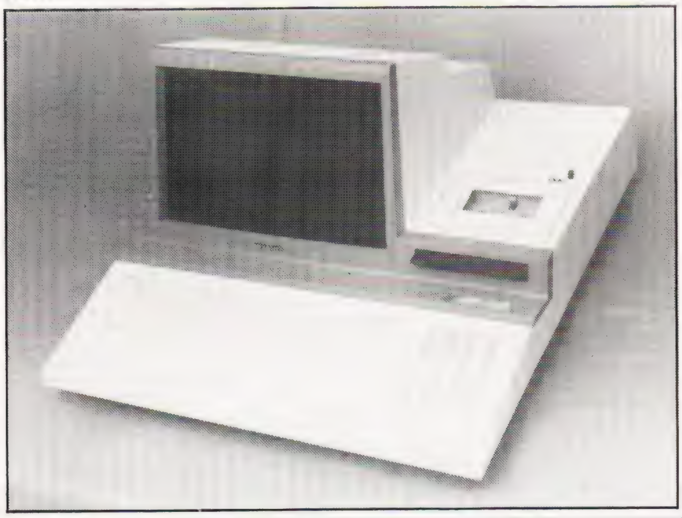
```
        ADC     LINPTR+1   ;Add in any carry.
        STA     (LINPTR),Y ;Put high byte back.
;
;Now run down the line pointers looking for the
;end of the program. (Two consecutive nulls in
;the line pointer locations).
;
MAINLP  LDY     #$00       ;Initialise index.
        LDA     (LINPTR),Y ;Get low pointer byte.
        BNE     NOTEND     ;Is it null?
        INY
        LDA     (LINPTR),Y ;Get high pointer byte.
        BNE     NOTEND     ;Is it null?
;
;End of program reached - copy address of the
;byte following the two nulls to VARTAB, ARYTAB
;and STREND.
;
        LDA     #$02       ;Step over two nulls.
        CLC                ;
        ADC     LINPTR     ;
        STA     VARPTR     ;Low pointer bytes.
        STA     ARYPTR     ;
        STA     STREND     ;
        LDA     #$0        ;Add in any carry.
        ADC     LINPTR+1   ;
        STA     VARTAB+1   ;High pointer bytes.
        STA     ARYTAB+1   ;
        STA     STREND+1   ;
        RTS                ;Routine finished.
;
;End of program not reached - read pointer to
;next program line.
;
NOTEND  LDY     #$00       ;Initialise index.
        LDA     (LINPTR),Y ;Get low pointer byte.
        TAX
        INY
        LDA     (LINPTR),Y ;Get high pointer byte.
        STA     LINPTR+1   ;Update LINPTR to
        STX     LINPTR     ;point to next line.
        SEC
        BCS     MAINLP     ;Repeat.
```

SHARP MICRODEALER

SHARP MZ-80A

MEMORY	48K RAM	4K ROM		
LANGUAGE	Microsoft BASIC			
CASSETTE	1200 baud (built-in)			
DISC	extra	DOS		
KEYBOARD	QWERTY ✓	CURSOR ✓	NUMERIC ✓	FUNCT ☐
DISPLAY	TV ☐	MONITOR ✓	SUPPLIED ✓	
INTERFACE	PARA ✓	SERIAL ☐	BUS ✓	
GRAPHICS	BLOCK ✓	USER ☐		
	LINE ☐	RES 80 by 50		
	COLOUR	TEXT 25 by 40		
SOUND	Single channel			

Notes: The Sharp MZ-80A is a Z80 based micro. An expansion unit printer, floppy disc unit and other peripherals are available. Other languages can also be used such as Pascal merely by replacing the tape. With the floppy disc option the machine can respond to higher level software such as Disc BASIC and FDOS (including BASIC compiler). A small range of business and educational software is available. The supplier is **Sharp Electronics (UK) Ltd,** Thorp Road, Newton Heath, Manchester M10 9BE



SHARP MZ-80B

MEMORY	64K RAM	2K ROM		
LANGUAGE	BASIC (on tape)			
CASSETTE	1800 baud built-in			
DISC	extra	DOS		
KEYBOARD	QWERTY ✓	CURSOR ✓	NUMERIC ✓	FUNCT ☐
DISPLAY	TV ☐	MONITOR ✓	SUPPLIED ✓	
INTERFACE	PARA ☐	SERIAL ☐	BUS ✓	
GRAPHICS	BLOCK ✓	USER ☐		
	LINE ✓	RES 320 by 200		
	COLOUR	TEXT 25 by 80		
SOUND	3 channels			

Notes: The Sharp MZ-80B is a Z80A based micro. Various other languages can be loaded as the machine is "soft", no language being fitted in ROM. Expansion unit, the MZ-80P5 printer and the MZ-80FB floppy disc drive are also available. The supplier is **Sharp Electronics (UK) Ltd,** Thorp Road, Newton Heath, Manchester.

BERKSHIRE

Kuma Computers Limited
Fully support with software and hardware. Sharp 700, Sharp MZ80A, and MZ80B.
11 York Road, Maidenhead, Berks. Phone for details — 0628 71778

COMPUTER 100
181 Oxford House, Reading, Berks RG1 7UZ
TEL: 0734 591616
One of the UKs largest stockists

CAMBRIDGESHIRE

MS CONSULTANTS (CAMBRIDGE) LTD
152 High Street, Huntingdon, PE18 6TF. Tel: 0480 51721
Specialists in Sharp Hardware & Software

LANCASHIRE

Sumita
12-14 Avenham St. Preston (0772) 51686
Sharp MZ-3541 Based Software, Bakehouse Orders System, Barstock System
SALES — SERVICE SOFTWARE

BAR INSTRUMENT CO. LTD.
For Sharp computers, books, software
Urmston Computer Centre, 124 Flixton Road, Urmston, Manchester.
Tel: 061 - 747 4626

MICRO APPLICATION PACKAGES LTD. Wild Oaks Drive, Thornton, Blackpool FY5 5BR. Tel: 0253 866464
Specialize in Business Programs for the Sharp MZ-3500 Machine. Estimating Job Costing. Retail Stock Control. Also Graphics Draughting Package
Suitable CAD CAM use

LINCOLNSHIRE

SHARP CENTRE
16 Melville Street, Lincoln.
Tel: Lincoln 32379
All Sharp Microcomputers. Specialists in Industrial Process Control Systems. Customers include British Telecom. M.O.D. CRA & CTA Members

LONDON

SHARPSOFT LTD.
With the Sharp user in mind. For the keenest prices on hardware or full range of software contact the Sharp Computer Specialists.
Sharpsoft Ltd, Crisallen House, 86-90 Paul Street, London EC2.
Tel: 01 - 729 5588.

MICROS FOR MANAGERS
149 Gloucester Rd., London SW7 4TH
Pocket Computer Software Specialists. APD air conditioning/heating/ventilation design package £25 (for Sharp PC1251 & PC1211). Applications written for you. Wholesale/Retail — Trade enquiries welcome. Tel: 01-370 5125 for appointment.

MIDLANDS

JAXREST LTD
Linton House, Catherine Street, Aston, Birmingham.
Tel: 021 328 0543
Sharp Service and Support

NORFOLK

GRASSROOT COMPUTERS
25 Wellington Rd., Dereham.
TEL: 0362 4640
Specialist in small business computing. Send for details of stockbox — the complete and thorough stock system with 2,500 items per disc. Other modules to follow.

SCOTLAND

micro change ltd
5 Annfield Place, Glasgow C31 2XN
Tel: 041 - 554 7623
SHARP ALTOS SUPERBRAIN
Call us for your software and hardware requirements — We will tailor our data base to meet these

LOWDATA SYSTEMS
A division of Lowdon Bros & Co (Engineers Ltd), PO Box 53, Blackness Road, Dundee, DD1 9JG
Tel: (0382) 22314
Full range of Sharp Equipment stocked

STAFFORDSHIRE

COMPUTER DISTRIBUTORS
DISCOS LTD.
East Cannock Road, Hednesford, Staffs WS11 5LT. Tel: Hednesford (05438) 2021-3
MZ 3541 Business Computers, Peachtree Business Software, Financial Director Accounts Package, MZ 700 Wholesalers

WALES

STEVE'S COMPUTER COMPANY LTD
CASTLE ARCADE, CARDIFF.
Tel: 0222 — 371578
OPEN: Mon — Sat. 9 - 5.30
For Components to Computers

WEST SUSSEX

MICROCENTRE LTD.
SYSTEMS, SOFTWARE SERVICE, SUPPORT
28-30 Station Road, Bognor Regis, Sussex.
TEL: 0243 827779

YORKSHIRE

PROGRAM 1
For Business Software
35 Albion Street, Hull.
TEL: 0482 20022

FOR DETAILS ON ADVERTISING IN MICRODEALER FEATURES — RING 01 - 437 0699

COMMODORE MICRODEALER

COMMODORE 720

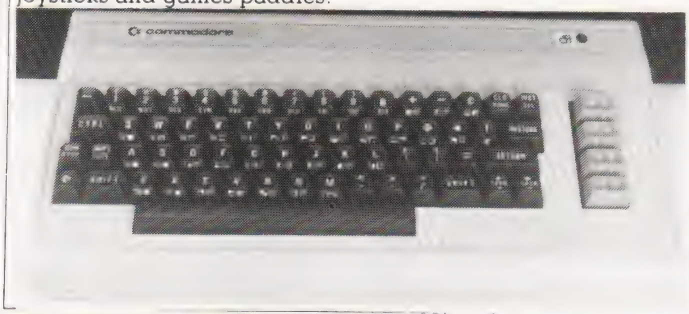
MEMORY	256K 20K ROM
LANGUAGE	Commodore BASIC
CASSETTE	300 baud
DISC	Twin in-built floppy drives
KEYBOARD	QWERTY ☑ CURSOR ☑ NUMERIC ☑ FUNCT ☑
DISPLAY	TV ☐ MONITOR SUPPLIED ☑
INTERFACE	PARA ☑ SERIAL ☑ BUS ☐
GRAPHICS	BLOCK ☑ USER ☐
	LINE ☐ RES 80 by 25
	COLOUR 16 TEXT 80 by 25
SOUND	**Three channels**

Notes. The Commodore 720 is the top model in the 700 range of business machines. It is built round the 6509 processor, but there is a dual processor (Z80 or 8088) option. The machine has been designed to meet the IEC specifications. The black-and-white monitor screen is integral and features tilt and swivel. The keyboard may be detached. The dual disc drives are built-in to the main housing and use DMA transfer, increasing speed.

COMMODORE 64

MEMORY	64K RAM 26K ROM
LANGUAGE	PET BASIC
CASSETTE	300 baud
DISC	extra DOS
KEYBOARD	QWERTY ☑ CURSOR ☑ NUMERIC ☐ FUNCT ☑
DISPLAY	TV ☑ MONITOR SUPPLIED ☐
INTERFACE	PARA ☑ SERIAL ☑ BUS ☑
GRAPHICS	BLOCK ☑ USER ☑
	LINE ☐ RES 80 by 25
	COLOUR 16 TEXT 40 by 25
SOUND	**Three channels**

Notes. The Commodore 64 is a 6510 based micro that can also use Pascal, COMAL, LOGO, FORTH and PILOT. Programs can be loaded from cassette recorder or disc drives, both extra, or cartridges. The various peripherals include printer, joysticks and games paddles.



LUCAS

LUCAS LX

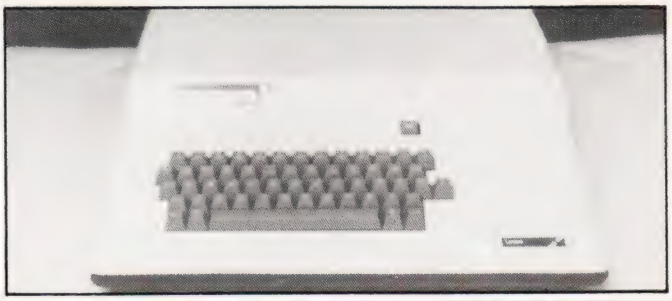
MEMORY	64K RAM expandable to 256K
LANGUAGE	Microsoft BASIC
CASSETTE	300 or 1200 baud
DISC	Single or twin 5¼ floppy disc drives DOS CP/M 2.2 (supplied) or NAS-DOS
KEYBOARD	QWERTY ☑ CURSOR ☑ NUMERIC ☑ FUNCT ☑
DISPLAY	TV ☑ MONITOR ☑ SUPPLIED ☑
INTERFACE	PARA ☑ SERIAL ☑ BUS ☑
GRAPHICS	BLOCK ☑ USER ☑ LINE ☐ RES 392 by 256 COLOUR 8 TEXT 80 by 25

Notes. The Lucas LX is a Z80 microcomputer aimed more at the professional and business user. Hence 5M Winchester disc interfacing is provided. Popular printers may be used with the RS232 serial interface, and a Centronics interface is also provided. There is an additional parallel interface connector for providing up to 16 on/off signals. The monitor supplied as standard is a 12" monochrome version: a colour monitor is also available. The high res colour graphics may be 392 by 256 in eight colours, or 784 by 256 in two colours. A wide range of applications software is available via the CP/M operating system, including Wordstar, Supercalc, and Calcstar.

NASCOM 3

MEMORY	48K RAM 10K ROM
LANGUAGE	Microsoft BASIC
CASSETTE	300 or 1200 baud
DISC	extra DOS CP/M or NAS-DOS
KEYBOARD	QWERTY ☑ CURSOR ☐ NUMERIC ☐ FUNCT ☐
DISPLAY	TV ☑ MONITOR ☑ SUPPLIED ☐
INTERFACE	PARA ☑ SERIAL ☑ BUS ☑
GRAPHICS	BLOCK ☑ USER ☑ LINE ☐ RES 800 by 256 COLOUR 8 TEXT 25 by 80
SOUND	Three channels

Notes. The Nascom 3 is a Z80 based micro. A second version of BASIC and Pascal are also available, as are a cassette recorder and light pen.



MICRODEALER

AVON

MicroValue
TARGET ELECTRONICS LTD
16 Cherry Lane, Bristol.
Tel: (0272) 421196
80-BUS SOLUTIONS

LONDON

MicroValue
HENRY'S RADIO
404 Edgware Road, London W2.
Tel: 01-402 6822
80-BUS SOLUTIONS

BUCKS

MicroValue
AMERSHAM COMPUTER CENTRE
18 Woodside Road, Amersham, Bucks.
Tel: (2403) 22307
80-BUS SOLUTIONS

NORTHERN IRELAND

NEWBURN
BALLYCARRY
CO ANTRIM
6 days until 8 pm. Gemini, Galaxy Nascom, Acorn Computers, Discs monitors, printers, servicing, hire Industrial Control, Accounts, Word Processors.
WHITEHEAD 78330

DEVON

PLYMOUTH'S DEALER nascom
S & R BREWSTER LIMITED
86-88 Union Street, Plymouth PL1 3HG
Tel: 0752 665011 Open: 6 days

NOTTINGHAMSHIRE

MicroValue
COMPUTERAMA, (Skytronics Ltd.)
357 Derby Road, Nottingham
Tel: (0602) 781742
80-BUS SOLUTIONS

DORSET

PARKSTONE COMPUTER SYSTEMS
Main Nascom Dealer & Service Centre
Also Dealers for Gemini, Transtec, Genie
18 Station Road, Lower Parkstone, Poole, Dorset BH14 8UB.
Tel: PARKSTONE (0202) 746555

ZIPPY ELECTRONICS
West Dorsets Nascom Dealer
Business and Personal Computer Systems, Printers, Software. Tel: Bridport (0308) 56539
Mail Order Service operated. Access/Barclaycard accepted.

SUFFOLK

M. D. W. ELECTRONICS
47 Woodbridge Rd. East, Ipswich
IP4 5QN.
Tel: (0473) 78295
Nascom, Nasbus, Gemini, Multi-board Stockists

SURREY

MicroValue
ELECTROVALUE LTD
28 St. Judes Rd, Englefield Gn, Egham, Surrey. Tel: (07843) 3603
80-BUS SOLUTIONS

HUMBERSIDE

TOMORROWS WORLD
The sole Nascom Agent for North Humberside
15 Paragon Street, Hull HU1 3NA.
TEL: 0482 24887

WALES

BYTE-RITE
Llandaff R & TV Ltd, 24-26 High St., Llandaff, Cardiff. Tel: 0222 563760
Nascom, Gemini, Lucas Logic, C.A.D. Networks, Dragon, Peripherals

LANCASHIRE

MicroValue
EV COMPUTING
700 Burnage Lane, Manchester M19.
Tel: 061-431 4866
80-BUS SOLUTIONS

WARWICKSHIRE

BUSINESS & LEISURE MICROCOMPUTERS
16 The Square, Kenilworth, Warwickshire CV8 1EB
Tel: 0926 512127
We are the largest Lucas Nascom dealer in the UK and carry the greatest variety of products for this range of computers.

LONDON

MicroValue
OFF RECORDS
Computer House, 58 Battersea Rise, Clapham Junction. Tel: 01-223 7730
80-BUS SOLUTIONS

YORKSHIRE

MicroValue
LEEDS COMPUTER CENTRE
55 Wade Lane, Merrion Centre, Leeds.
Tel: (0532) 458877
80-BUS SOLUTIONS

SEE YOUR BUSINESS GROW
PHONE ASP LTD ON
01 - 437 0699

COMPUTING TODAY

Lineage: 35p per word.
Semi display: £8.00 per single column centimetre
Ring for information on series bookings/discounts.

All advertisements in this section must be prepaid.
Advertisements are accepted subject to the terms and conditions printed on the advertisement rate card (available on request).

01-437 0699

Send your requirements to:
MARK BECAREVIC
ASP LTD, 1 GOLDEN SQUARE, LONDON W1

SOFTWARE APPLICATIONS

48K SPECTRUM & BBC

POOLS PREDICTOR

An easy to use and very powerful forecasting program using 6 different methods of prediction, based on analysis of current form. **£4.99**

MAYDAY SOFTWARE
Dept CT, 181 Portland Crescent, Stanmore, Middlesex HA7 1LR

CORTEX — FORTH

For fig-forth with extensions for Cortex Colour Computer. 16K supplied in two 2564 Eproms. Replaces 1st two Basic Eproms.

£35 INCLUSIVE

LOMBARD SYSTEMS
18 Lombard Street, Lidlington, Bedford MK43 0RP.

TRS-80 MODEL I
1500 BAUD CASSETTE SPEED-UP UTILITY

Load and Save Programs at 3 times normal speed!

- No hardware modes needed
- Works on BASIC & SYSTEM programs
- Displays program as it loads
- Programs can be up to 14.5K long (16K Level II only)

£7.50 inc p&p to:

D JACKSON (Software)
37 Sandhurst Road, Bexley, Kent.
Tel: 01 - 304 3991

DRAGON ADVANCED PROGRAMMER'S PACKAGE

The most comprehensive FORTH implementation on the market, suitable for 32K and 64K. Includes the language itself which can access Basic commands, powerful text editor, 6809 macroassembler, extensive documentation and the fig-FORTH Installation Manual. **Plus:** Dragon Companion Book containing full 6809 dissassembler, extensive memory map, many tips to get the most from your Dragon. Excellent value at only **£15** inc.
M & J Software, 34 Grays Close, Scholar Green, Stoke-on-Trent. (0782) 517876

SOFTWARE LANGUAGES

BackFORTH

FORTH for **NASCOM's**. 32K min. written in Z80 code (NASCOM 1 with NAS-SYS and Cottis Blandford). Supplied as a tape based system (pseudo disc in RAM).

BackFORTH features include:

Editor, screen and line based modes. Versatile debugger to enable single stepping through FORTH secondaries or machine code. Macro assembler. Z80 style mnemonics and high level constructs. 32 bit integer extension. Hashed directory, enables fast searches for the 380+ words. Interfaces to NAS-SYS routines. The documentation (approx 150 pages) includes glossary, source listings, memory map and definitions to provide figFORTH simulation. Available now at **£28.75** inclusive or send SAE for further details.

Dept CT1, Southdown Computer Services, Manresa, Cooksbridge, Lewes, East Sussex BN8 4SP.

SOFTWARE GAMES

GROVE SOFTWARE

Games for the Jupiter Ace with 16K

TBT 1 Mastermind + Bomber
TBT 2 Pairs + Submarines

£5.50 each inc p&p

SAE for Catalogue
Cheques and PO's to

GROVE SOFTWARE
2 Grove Court, Oakfield Road, Penge, London SE20 8RG.

ATOM OWNERS BEWARE. The swarm approaches. Destroy their dynamic waves or be doomed. Hall of Fame, sound, m/c 12K. £4.95. M.J.R. Software, 4 Fulbeck Avenue, Leicester.

SOFTWARE EDUCATIONAL

ISIS VIDEO

Specialists in Educational Software, Software and Textbooks/Software Packages from ABC to A-Level. Many major publishing houses included in our comprehensive catalogue covering most popular home and school computers. Ring Fiona Brooks now on 01 - 549 9305, or write stating model to **Isis Video (Dept CT), Crown Works, Church Road, Norbiton, Kingston, Surrey KT1 30B**

ACCESSORIES

CHILTERN ELECTRONICS

Amazing Value in Video Monitors

Professional quality equipment by AGC Corp ideal for your micro. Look at these features ★ Supero resolution for graphics 80 col. etc. ★ Bandwidth 10hz-22Mhz ★ Green Screen P31 phorsphor ★ Antiglare tube ★ Attractively styled case ★ Composite video mains operation. Available in 9" Model HM911 **£85.** 12" Model HM123 **£89.**

FANTASTIC KEYBOARD BARGAIN!! Scoop purchase allows us to offer a 49 key QWERTY layout, ASCII encoded keyboard. Fully cased, used but guaranteed working. Full logic diagrams and information supplied. **ONLY £15.** Add carriage £1.85 and VAT. Educational discounts. Access welcome. Same day despatch. Call Gary Kent on 02407 71234. **High Street, Chalfont St Giles, Bucks HP8 4HQ. Telex: 262284.**

BLANK CASSETTES!

TOP QUALITY PROFESSIONAL BRAND COMPUTER/AUDIO CASSETTES AT BUDGET PRICES

Packed in boxes of 10 cassettes complete with labels, inlay cards and library cases. Prices include VAT post & packing.

Length	Box Price (10)	Qty	Amount
5 min	£4.85		
10 min	£4.90		
12 min	£4.95		
15 min	£5.00		
30 min	£5.20		
60 min	£5.80		
90 min	£7.50		

Cheque/Postal Order enclosed for £
NAME ....
ADDRESS ....

PROFESSIONAL MAGNETICS LTD
Cassette House, 329 Hunslet Rd, Leeds.
Tel: (0532) 706066
TRADE ENQUIRIES WELCOME

HARDWARE

NEWBRAIN

The professional micro computer for the price of the hobby machine. Delivery free. Call us for very special offer. Mail Order. Access.
Call **Stevenage (0438 812439)**
Anytime for the latest details of Newbrain Hardware, Software, Books. "Newbrain Dissected" Book **£9.00**, tape **£9.95** and both **£18.45**. "Getting More from your Newbrain" **£7.50**. All include p/p. Epson, CPAT, Juki & KDC printers, monitors & recorders. Chess **£14.95**.

ANGELA ENTERPRISES
4 Ninnings Lane, Rabley Heath, Welwn, Herts AL6 9TD.

FUTURE TECHNOLOGIES

(ETI) 1 Hollybrook Road, Shirley, Southampton SO1 6RB

CBM 64 Computer **£198**
80 Column Board for 64 .... **£115**
with cassette & 2 blank tapes **£242**
VIC 20 starter pack **£139**
Cassette alone **£44.90** Tapes **£1 ea**
Spectrum 16K **£99.95** 48K **£129.95**
Also Atari, Texas, accessories, software. Cheque/PO with order
Send large SAE for lists (23p stamp)
Ad £1 P&P for orders under £50
All prices include VAT

SERVICES

ELIMINATE FAULTY CASSETTES

DataClone is the first company in the UK established specifically for the duplication of data cassettes.

All other duplicating houses are audio orientated — only DataClone has a duplicating system designed from scratch purely to handle computer information.

The result?

Greatly improved reliability in data transfer rates from 300 to beyond 2400 baud — previously unattainable.

All formats catered for. Quantities from 100 to infinity.

Contact us now for brochure.

DataClone — the first specialist service for computer cassettes.

DATACLONE
Unit 1, Roslin Square, Roslin Rd., Acton, London W3.
Tel: 01-723 0477 Telex: 21879

WHICH COMPUTER? Hobbyist or Business. For individual assessment from non-dealer forward £3 together with name and address to: C & S Advisory Services, 144 Sutcliffe Avenue, Grimsby, South Humberside DN33 1AP.

FOR SALE

SPECTRUM 48K. Brand new plus five games cassettes. £115. Tel: evenings 061 748 2175.

LIBRARIES

THE MZ80K/A SOFTWARE LIBRARY

Choose from 150 titles. These include exciting Arcade/Adventure games, and Educational/Business applications. Once only membership fee **£5.00**. Hire charges only **90p** per 14 day period. Plus your first 5 games FREE. Send now for free illustrated catalogue to:

The Yorkshire Software Library, 13 Park Top, Pudsey, Yorkshire LS28 6BY.

CARTRIDGE CITY will rent Atari 400 and 800 owners cartridges at low rates. Discounts on software purchases. Details from Cartridge City, 25 Gaitside Drive, Aberdeen AB1 7BH.

FOR SALE

NASCOM-2 NAS-SYS-3, 32K RAM, NAS-DIS NAS-ASS NAS-PEN NAS-DRAW 8K Basic graphics, PSU, full documenta-tion tapes. £330 ono. Phone evenings 0933 678421.

SHARP MZ80K software for sale. Worth £100 plus. Only £30 the lot Phone 01-472 1331 for details.

TANGERINE MICROTAN, Tanex, hi-res graphics board, Basic, X Bug, keyboard, PSU £140 ono. Ramsey (Cambs) 831552.

Hard disc drive symbiotic 5 megabite Winchester plus Apple II interface formatted. Offers on **£950**.

TEL: Weymouth (03057) 74447

NASCOM 1, Hobbit micro-cassette deck, 48K, Zeap, graphics, Eprom programmer etc. £195 ono. Macclesfield (0625) 72988 evenings.

FOR MEN! VASECTOMY FREE DETAILS

Everlasting alternative to the Pill.

No fuss. No waiting lists. One visit. Low cost. In

BIRMINGHAM	LONDON	PLYMOUTH
GLOUCESTER	MANCHESTER	SOUTHAMPTON
LEEDS	NEWCASTLE	SWANSEA
LIVERPOOL	NORWICH	

Write today. Booklet by return. Plain cover. Or phone

LONDON	**01-388 2585**
LEEDS	**0532 440685**
MANCHESTER	**061 832 4260**

Name ____________

Address ____________

CT1

Marie Stopes House, 108 Whitfield St., London W1P 6BE
Caring clinics since 1925

MARIE STOPES SEND NOW

ADD-ONS

48K and 16K SPECTRUM EPSOM CARD (4K)

The Eprom Card uses spare ROM space, so normal operation of Spectrum is not affected. Prices: Edge Connector Version (plugs directly into Spectrum £21.25. Motherboard Version £18.50. CTR Motor Controller Option - £3.45. Plug in RAM - Adaptor £9.95. Programmers Eprom (£9.95). Ten Top Quality, User-Friendly Routines. Right & Left Scroll, Size, Memory, Renumber (inc. GOTO, GOSUB etc). Variables Dump, Word processor, Block Delete, Sprite (Universal Character Printer), Graphics Set. New! ZX Spectrum (Eprom Programmer £28.75) and 25 volt P.S.U. (£7.75). Dissassembler Eprom (£9.95). Also available ZX81 Epsom Card, Programmer and Toolkit Eprom. SAE please for extra details. P&P 75p (UK) £2.00 (overseas). **Orme Electronics, 2 Barripper Road, Camborne, Cornwall. Tel: 0209 715034 or 087 257 2842**

CLASSIFIED ADVERTISEMENT — ORDER FORM

1.	2.	3.
4.	5.	6.
7.	8.	9.
10.	11.	12.
13.	14.	15.

When placing your ad, please state classification required. 35p per word.
Send to: **ASP Classified, 1 Golden Square, London W1.**

Tel: 01 - 437 0699

Name . . . .

Address . . . .

. . . .

. . . .

Tel.No.(Day) . . . .

Please place my advert in **COMPUTING TODAY** for issues commencing as soon as possible

THE LONDON HOME COMPUTER SHOW

AB & C Computers

MAIL ORDER SPECIALISTS

SEND LARGE SAE FOR LISTS INSTANT CREDIT FOR CALLERS

A SELECTION FROM OUR EXTENSIVE STOCK

★TEXAS SPECIAL OFFER PACKAGE★

T199 4A Computer + Extended Basic including Delivery **£165.95** whilst stocks last.

Titch Software for T199/4A Hangman, Victory 4. Super Value at **£8.00**

Protek Spectrum Joystick Interface **£14.95** inc. P&P. Uses most switch type Joysticks.

New Oric Joystick Interface **£14.95** inc. P&P

New MCP-40 Centronic 4 Coloured Printer fits most personal Micros **£149.95** inc. P&P

ZX Spectrum & BBC Upgrades

Now in stock DRAGON DISC DRIVES £275.00 + £5. Delivery

New low prices on Texas TI99/4A. Contact us first.

Are you short of a cassette recorder for Xmas — for your computer CONTACT US.

Send Cheque with order made payable to AB & C Computers
Duchy House, 6 Lower Aylmer Square, St Austell, Cornwall.

COMPUTAMART

AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...

COMPUTAMART

AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...AT A GLANCE...